Business Intelligence

Vol. 4, No. 12

CUTTER CONSORTIUM

Agile Data Warehousing:

Incorporating Agile Principles

by Dr. Ken Collier, Senior Consultant, Cutter Consortium; with Jim Highsmith, Director, Cutter Agile Project Management Practice

This *Executive Report* targets organizations that are considering a data warehousing project, organizations that have struggled to implement a data warehouse, data warehouse developers, and IT executives who are responsible for overseeing such projects. The report recommends a leaner approach than that of traditional methods; it advocates a working system that meets users' business needs rather than heavily process-centric development approaches that emphasize documentation and contractual agreements. Armed with the agile data warehousing approach, organizations can increase the likelihood of a successful data warehouse implementation on time and within budget.

Executive Report

Cutter Business Technology Council

About Cutter Consortium

Cutter Consortium is a truly unique IT advisory firm, comprising a group of more than 100 internationally recognized experts who have come together to offer content, consulting, and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and agile project management, enterprise architecture, business technology trends and strategies, enterprise risk management, metrics, and sourcing.

Cutter offers a different value proposition than other IT research firms: We give you Access to the Experts. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts; experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats including content via online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products and training/consulting services, you get the solutions you need, while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business-technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

Agile Data Warehousing:
Incorporating Agile Principles

BUSINESS INTELLIGENCE ADVISORY SERVICE

Executive Report, Vol. 4, No. 12

by Dr. Ken Collier, Senior Consultant, Cutter Consortium; with Jim Highsmith, Director, Cutter Agile Project Management Practice

INTRODUCTION

This *Executive Report* targets organizations considering a data warehousing project, organizations that have struggled to get a data warehouse implemented, data warehouse developers, and IT executives whose job it is to oversee such projects. By combining experience in data warehousing (Dr. Ken Collier) and in agile development practices (Jim Highsmith), the authors have developed a new approach to data warehousing projects called agile data warehousing (ADW).

While the ADW method incorporates the tried-and-tested fundamentals of data warehousing, it adapts many of the successful agile software development practices and principles to data warehousing techniques. The report doesn't suggest a revolutionary change in data architectures, modeling methods, or warehousing technologies; instead it recommends a lean approach that emphasizes a working system that meets users' business needs rather than heavily process-centric development approaches that emphasize documentation and contractual agreements.

The goal of this report is to change the ways that data warehousing projects are implemented. Data warehousing is difficult, and there are many accounts of failed projects. Our experience suggests that an ADW approach will greatly increase the likelihood of successful implementation of a data warehouse on time and within budget.

This approach has strong roots in the work of AgileAlliance members. We have adapted published techniques such as Test-Driven Development (TDD) from Cutter Consortium Senior Consultant Kent Beck, the Framework for Integrated Test (FIT) from Ward Cunningham, Agile Modeling from Cutter Consortium Senior Consultant Scott Ambler, and others to the unique characteristics of data warehouse development. We have also incorporated experience and lessons learned from real ADW projects and the work of other experts.

WHEN DATA WAREHOUSING GOES BAD

Traditional data warehouse projects follow a typical waterfall development model in which rigorous efforts are made to collect complete requirements, design comprehensive architectures and data models, develop and populate repositories, and, ultimately, develop the analytical reports and artifacts that users want. These projects are complex affairs, involving a project manager leading a team of specialists including business analysts, data architects, and so forth. Depending on their magnitude, these projects generally run at least six months and can easily exceed US $1 million.

I have worked with some talented and experienced data warehouse developers. I've also had the benefit of working with savvy clients who have reasonable expectations and a pretty stable set of requirements. Sounds like a recipe for total success, right? But often, business users are less than ecstatic about the first data warehouse production rollout. User reactions commonly range from "This isn't what I was told I would get" to "I can see where this might be useful with some refinement."

Most data warehouse developers have participated in projects that were less than successful. I recently worked with a midsize company seeking to replace its existing homegrown reporting application with a properly architected data warehouse. At the outset, the project seemed poised for success and user satisfaction. Despite the best efforts of developers, project managers, and stakeholders, however, the project ran over budget and past deadline, and users were less than thrilled with the outcome. Because this project in particular motivated the development of ADW, the following offers a brief retrospective to highlight the principles and practices presented later in this report.

Project Attributes

Existing application. Internally, the company's existing reporting application was referred to as a "data warehouse." In reality, though, the data model was a replication of parts of the legacy operational database. This replicated database did not include data scrubbing and was wrapped in a significant amount of custom Java code to produce the specified reports required. At various times, users had requested new custom reports, thus overburdening the application with highly specialized and seldom-used reporting features. All the reports could be classified as canned reports. No advanced analytical capabilities were provided.

Project motivation. Because the existing "data warehouse" was not architected according to data warehousing best practices, it had reached the practical limits of maintainability and scalability needed to continue meeting user requirements. Additionally, with the new billing system coming online, it was evident that the existing system could not easily be adapted to accommodate the new data. Therefore, at the executive level, there was strong support for a properly designed data warehouse.

External drivers. A sales team from a leading worldwide vendor of data warehousing and business intelligence (BI) software initially envisioned the data warehousing project. Providing guidance and pre-sales support, the sales team helped project sponsors understand the value of eliciting the help of experienced BI consultants with knowledge of industry best practices. But as with many sales efforts, initial estimates of project scope, cost, and schedule were too ambitious.

Development team. The development team comprised exclusively external data warehousing consultants. Because the

The ***Business Intelligence Advisory Service Executive Report*** is published by Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Client Services: Tel: +1 781 641 9876 or, within North America, +1 800 492 1650; Fax: +1 781 648 1950 or, within North America, +1 800 888 1816; E-mail: service@cutter.com; Web site: www.cutter.com. Group Publisher: Kara Letourneau, E-mail: kletourneau@cutter.com. Production Editor: Linda M. Dias, E-mail: ldias@cutter.com. ISSN: 1540-7403. ©2004 by Cutter Consortium. All rights reserved. Unauthorized reproduction in any form, including photocopying, faxing, and image scanning, is against the law. Reprints make an excellent training tool. For information about reprints and/or back issues, call +1 781 648 8700 or e-mail service@cutter.com.

company's existing IT staff had other high-priority responsibilities, the development team did not have extensive knowledge of the business or existing operational systems. The development team did, however, have open access to business and technical experts within the company as well as technology experts from the company's software vendor. While initial discovery efforts were challenging, all stakeholders participated extensively.

Customer role. The company's finance division filled the role of the primary "customer" for the new data warehouse, and the CFO sponsored the project. Together, the customer team and the CFO had a relatively focused business goal of gaining more reliable access to revenue and profitability information. They also had a substantial amount of existing reports used in business analysis on a routine basis, offering a reasonable basis for requirements analysis.

Project management. Corporate IT handled project manager responsibilities using traditional Project Management Institute (PMI) practices. The IT group was simultaneously involved in two other large development projects, both of which had direct or indirect impact on the scope of the data warehouse project.

Hosted environment. Due to limited resources and infrastructure, the company's IT leadership had recently decided to partner with an ASP to provide hosting services for newly developed production systems. The data warehouse was expected to reside at the hosting facility located on the West Coast of the US, while company headquarters were located on the East Coast. Because legacy systems remained on the corporate infrastructure, separate geographic locations for the warehouse and headquarters created complications — though not insurmountable ones — for the movement of large volumes of data.

Project Outcome

The original project plan called for an initial data warehouse launch within 90 days. As any experienced data warehouse developer knows, such a deadline imposes an ambitious, but not impossible, schedule (assuming appropriate scope definition). But the data warehouse launch for this project came a full eight months after project start. User acceptance testing did not go well. Users were already annoyed with project delays, and when they finally saw the promised features, there was a significant gap between user expectations and the end product. As is common with late projects, staff members were added to the development team midstream to try and get the project back on track. So project costs far exceeded the planned budget, and the project was placed on hold until further planning could be done to justify continued development.

Retrospective

So who was to blame? Users thought that developers had missed the mark and failed to implement all their requirements. Developers believed that user expectations had not been properly managed and thus project scope had grown out of control. Project sponsors thought that the vendor and the consulting firm had overpromised and underdelivered. The vendor and consulting firm believed that internal politics and organizational issues were to blame. Finally, many members of the company's IT staff felt threatened by their own lack of ownership on the project and secretly celebrated the failure.

The project degenerated into a series of meetings to review contracts and project documents to determine who should be held responsible. And guess what? Everyone involved was partially to blame. In addition to the common technical challenges of data extraction, integration, and cleansing, the following were identified as root causes of project failure:

- The project contract did not sufficiently define the project scope.
- Requirements were incomplete, vague, and open-ended.
- There were conflicting interpretations of the previously

approved requirements and design documents.

- Developers put in long nights and weekends in a chaotic attempt to respond to user changes and new demands.
- Developers did not fully understand user requirements or expectations and did not manage requirements changes well.
- Users had significant misconceptions about the purpose of a data warehouse since the prevailing understanding of a warehouse was based on the previous reporting application (which was not a good model).
- Vendors made ambitious promises that developers could not fulfill in the time available.
- The project manager did not manage user expectations.
- IT staff withheld important information from developers.
- The ASP partner did not provide the level of connectivity and technical support that developers expected.

Hindsight truly is 20/20, and in the waning days of this project, several realities became apparent:

1. A higher degree of *interaction* between developers, users, stakeholders, and internal IT experts would have ensured accurate understanding between all participants.
2. Early and frequent *working software*, no matter how simplistic, would have greatly reduced users' misconceptions and increased the accuracy of their expectations.
3. Greater emphasis on *user collaboration* would have helped to avoid conflicting interpretations of requirements.
4. A project plan that focused on *adapting to changes* rather than on meeting a fixed and arbitrary deadline would have greatly improved user satisfaction with the end product.

Regardless of who is to blame, the root cause of data warehousing project failure is a disconnect between user and developer expectations.

In the end, and regardless of who is to blame, for this data warehousing project and many other failures, the root cause is a disconnect between user and developer expectations.

ADW BACKGROUND (Ken Collier)

Until 2003, upon meeting AgileAglliance cofounder Jim Highsmith, I was only superficially aware of the Alliance, a group of thought leaders from the software engineering community. I had focused on advancements in data warehousing and BI rather than on software engineering. But after meeting Jim, I began to see the connection between agile principles and data warehousing practices.

Jim and I, along with another colleague, met weekly to share coffee, experiences, and ideas. Jim would talk about his Agile Project Management (APM) principles as they unfolded, and I would lament my most recent data warehouse project for being understaffed and overscoped and for having ever-changing user requirements. My development team would work overtime, and yet users were never satisfied.

But during a weekly meeting, I had an “aha!” experience: it dawned on me that all this agile stuff that Jim and others have talked and written about has a direct application to data warehousing and BI in general. I realized it made sense to establish a set of values, principles, and practices for infusing agility into all BI projects, from canned reporting to data visualization, data mining and quantitative analytics, and of course, data warehousing. I have since put ADW principles into practice and am enthusiastic about the results.

Using ADW practices, I was able to lead three developers with little experience in data warehousing to complete a data warehouse in two and a half months without our team having to pull a single all-nighter. Although our resulting data warehouse will mature over time, the complete end-to-end architecture has been established, and multiple OLAP reports are available for use. Author Luke Hohmann likens early-stage agile projects to babies: they are complete but immature. This is a useful way to think about ADW. Our goal is to get a working system in the hands of users as early as possible so that we can start getting feedback and improving the system.

In the spirit of full disclosure, ADW borrows unabashedly from the works of agile pioneers such as Ambler, Beck, Martin Fowler, Highsmith, Hohmann, Cutter Consortium Senior Consultants Alistair Cockburn and Ken Schwaber, and others. My contribution is the adaptation of these thought leaders' ideas to industry-standard data warehousing best practices. Using my own knowledge and experience, and with much coaching from Jim, my aim is to establish a practical, applicable, and more successful data warehouse development alternative.

ADW AND APM (Jim Highsmith)

Ken and I met less than two years ago and were introduced by a mutual friend. Many of our early conversations concerned politics, restaurants, and especially skiing, but eventually discussion turned to the project Ken was working on and my work in agile software development and project management. As Ken talked about his data warehousing project and others that exhibited similar waterfall characteristics, we discussed how an agile approach might have mitigated some of the problems.

Since then, and as discussed in this report, Ken began to use agile practices — and in particular, APM practices — with successful results. We have also worked together implementing agile development and project management with a joint client on a data warehouse–like project. As I learned more about data warehousing and BI from Ken, and he learned more about agile development from me, we came to the mutual conclusion that agile practices could solve some of the ills plaguing traditional data warehouse and BI projects.

Historically, these projects are the epitome of big design up front. Even worse, the usual implementation sequence for these projects leaves user reporting until late in the project — after all the data staging, cleansing, reporting data base development, and other "technically" challenging portions of the project are complete. In many cases, unfortunately, when users finally see the results, they don't like what they see. Doubly unfortunate, by the time users see the results that fall short of their expectations, most of the project's resources have been spent and options have become limited. With the noose on the project tightening, the blame game begins, because everyone has already lost.

Ken and I believe that combining agile practices with data warehouse development has tremendous potential to ensure that these projects deliver value to customers early and often in a project. In place of a huge project that goes on for 12 to 18 months before users have access to valuable results, this approach has the potential to deliver results in just a few months, with subsequent quarterly deliveries.

Note: Ken has authored the bulk of this report; I've chipped in here and there, especially in the APM section below.

APM

An Agile Synopsis

In 2001, the AgileAlliance outlined a set of core values, principles, and practices for developing software that better meets users' needs and expectations. The most significant outgrowth of the first AgileAlliance workshop in 2001 was a statement of shared development values called the Manifesto for Agile Software Development [2], which states:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation

> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right [roman typeface], we value the items on the left [bold typeface] more. [2]

For practitioners like me with technical roots in the rigors of prescriptive methodologies, the Agile Manifesto is as powerful as it is simplistic. After years of practicing "heavyweight process" development, I find these agile values liberating. In my experience, when push comes to shove, we instinctively adopt the values expressed in the Agile Manifesto. Unfortunately, when push comes to shove, we have often crossed the boundary between order and chaos. By adopting these values from the outset of a project, we can maintain order from the beginning and gain satisfaction in the knowledge that our time is spent productively on creating working software.

From Agile Principles to Business Objectives

APM is characterized by envisioning and exploring processes rather than planning and doing processes. APM addresses projects in which the solution is unknown in the beginning and the project team must explore the problem and solution space to create a product that meets the customer's vision and delivers customer value. When a project has significant risk and uncertainty, it requires an envision→explore process. Exploration includes experimenting with different solutions to see which ones work.

Early on in high exploration–factor projects, the team may not know the complete feature list. It may have a reasonable product vision, various business constraints (e.g., target schedule, target costs), a general idea of key features, and some ideas about the overall architectural skeleton, but the details will emerge as the team delivers completed features every couple of weeks. The team's process is one of evolution and adaptation, not planning and optimization; the process embraces change.

While most people recognize industry volatility, they have not truly modified either their mindset or their management practices to acknowledge change. For example, the perceived high cost of change has driven methodologies to strive to limit change by engaging in extensive front-end planning and analysis. The problem is that no matter how thorough this early work, stuff happens; change happens. So rather than allow the high cost of change to dictate process, agile developers focus on reducing costs; low-cost change enables a development cycle of minimum up-front work followed by a succession of short development iterations. When we adequately reduce the cost of change, the entire economics of software development changes; the process shifts from one based on anticipation (define, design, and build) to one based on adaptation (envision, explore, and refine).

While plan→do processes work for low exploration–factor projects, they are no match for today's volatile business environment. So software development and project management processes must be geared toward mobility, experimentation, and speed. But first, they must be geared toward business objectives.

Reliable Innovation

There are five key business objectives for agile software development and APM: (1) continuous innovation (delivering on current customer requirements); (2) product adaptability (delivering on future customer requirements); (3) reduced delivery schedules (meeting market windows and improving ROI); (4) people and process adaptability (responding rapidly to product and business change); and (5) reliable results (supporting business growth and profitability).

Continuous Innovation

Developing new software applications, such as those for data warehousing and BI, requires a mindset that fosters innovation.

Business value isn't derived from these applications by following prescribed paths but by blazing new trails.

Product Adaptability

No matter how well a data warehouse application is developed, the future always brings surprises. As an enterprise changes, as new executives use the application, as business processes evolve, as new markets are pursued, the needs of data warehouse and BI applications change as well. The only way to survive is to strive for adaptability — a critical design criterion for a data warehouse application. Agile technical practices focus on reducing the cost of change (or adaptation) as an integral part of the development process.

Reduced Delivery Schedules

Reducing delivery schedules remains a high-priority business goal for executives, but providing incremental business value is just as important. Would you rather have a complete data warehouse application in 18 months or incremental data marts every three months? Further, incremental delivery can significantly increase the ROI of projects as well as generate valuable feedback from the development process through incremental releases. APM contributes to reducing delivery schedules in two key ways: focus and streamlining.

While many organizations strive for standardization of process and practices, they should strive for consistency.

First, the constant focus on features and the priority for their release in short, iterative timeboxes forces teams (comprising customers and developers) to consider both the number of features to include and the depth of those features. This reduces the overall workload by eliminating marginally beneficial features. Second, APM streamlines the development process by concentrating on value-adding activities and eliminating overhead.

People and Process Adaptability

Have you ever worked on two projects that were identical: that is, they shared the same people, same problem, same constraints, same customers, and the same executives? Why, then, do many organizations insist that the same processes and practices should apply to every project? While many organizations strive for standardization of process and practices, they should strive for consistency. What we need in most instances is a consistent framework with local variations in process and practices to accommodate differences among projects. We must also build adaptable teams whose members are comfortable with change and view change as integral to thriving in a dynamic project environment rather than as an obstacle.

Reliable Results

Manufacturing processes are designed to be repeatable and to deliver the same result time after time. They are predictable and therefore repeatable. However, because of the uncertainty and risk, exploration processes are different. They can deliver on a customer's vision and according to a customer's requirements as these requirements evolve, but they can't deliver a completely prespecified result. APM delivers consistent, reliable results.

The APM Framework

As shown in Figure 1, the APM framework supports the business objectives of continuous innovation, product adaptability, reduced delivery schedules, people and process adaptability, and reliable results through its five phases: envision, speculate, explore, adapt, and close.

In practice, projects have two iterative cycles of collaborative planning and collaborative development, as shown in Figures 2 and 3. In APM, as in all agile methods, both planning and development are done collaboratively. The collaborative planning cycle, which focuses on product vision, project scope and boundaries, and the overall project release plan, can — and should — be used throughout the project when

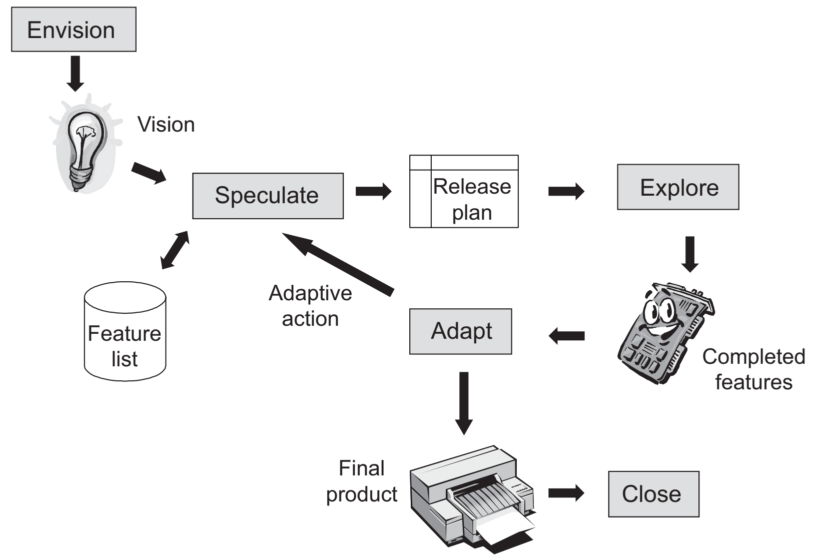

Figure 1 — Highsmith's Agile Project Management framework. (Source: [7].)

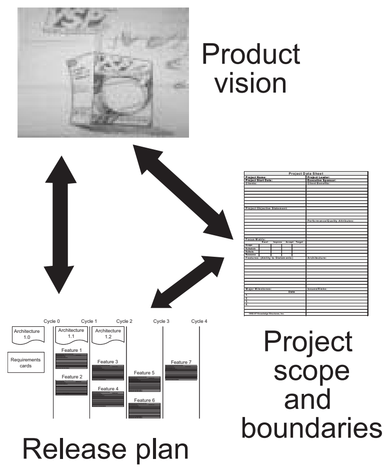

Figure 2 — The collaborative envision cycle. (Source: [7].)

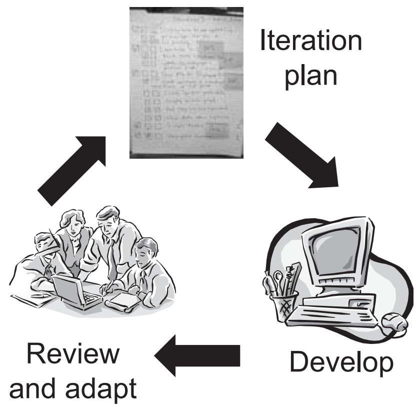

Figure 3 — The collaborative development cycle. (Source: [7].)

enough new information has been gathered to significantly alter the planning effort. The collaborative development cycle defines what is accomplished during each two-to-four-week iteration: detailed planning, development, and adaptation.

Envision

During the envision phase, customers and the project team create a product and project community vision that covers who, what, and how. Visioning is visual; the team must create a picture that focuses developers and customers on the critical aspects of the product. One technique for doing so is the product vision box. The team designs a "box" in which the application will be packaged, which forces the team to focus on exactly who the customer is and the key three or four features that will sell the application. The second aspect of visioning is to create a picture of the entire project community — of customers, product managers, project team members, and stakeholders — and *how* these parties intend to work together. Other specific practices in the envision phase include (1) generating product architecture and guiding principles, (2) creating the project data sheet, and (3) process and practice tailoring.

Speculate

While the word "speculate" may create an image of reckless risk taking, one dictionary definition is "to conjecture something based on incomplete facts or information." As mentioned previously, APM builds on an envision→explore rather than a plan→do model, and therefore the implied certainty usually attributed to the word "plan" must be overcome. APM may reliably deliver value, but recognizing that value may require a few twists and turns.

The two critical characteristics of the release-milestone-iteration planning practice used in this phase are short, timeboxed iterations and feature-based planning. Iterations are generally two-to-six-week timeboxes. Features, not tasks, are assigned to each iteration. Features are represented by index cards on which sufficient information is recorded to conduct the planning. At the end of these iterations, features are complete only if they are "done-done": that is, if they have been developed and unit-tested (done) and then user acceptance–tested (done-done). If eight features are completed during an iteration, the velocity of development is eight features per iteration, and this delivery rate is used for planning the next iteration. The two criteria for assigning features to iterations are (1) delivering customer valuable features (as prioritized by the customer representatives on the team) and (2) implementing features that reduce project risk.

As shown in Figures 2 and 3, speculating is done at two levels: for the entire project and for the next iteration. The overall envision planning cycle may be repeated every two to six iterations depending on how much has changed since completion of the previous release plan. Specific practices in this phase include creating the product feature list, feature (or story) cards, and the release-milestone-iteration plans.

Explore

The explore phase develops product features. During this phase, the project team delivers the features identified in the release and iteration plans, and project managers must focus on three critical activity areas: (1) delivering features by managing the workload and using appropriate technical practices; (2) creating a collaborative, self-organizing project community; and (3) managing the interactions among developers, customers, executives, and other stakeholders.

Projects have a beginning and an end, and each must be recognized. The failure to identify a project's end point can create problems of perception among customers.

In the explore phase, specific practices include workload management (the team manages the distribution of its work); technical practices, such as ruthless testing and refactoring; coaching and team development; and participatory decision making.

Adapt

Project managers usually refer to the adapt phase as monitoring and correcting and to the results of this activity as "corrective action." The phrase "corrective action," which actually means to correct according to the plan, implies that the plan was correct and that the reason for variation from the plan is poor team performance. APM uses the term "adaptive action," which implies something quite different. Adaptive action implies modification or change rather than success or failure. Agile projects are guided by the philosophy that responding to change is more important than following a plan; so while team performance may be an issue, the change may be attributable to poor planning or new information.

In an agile project, every iteration's conclusion provides a forum for the team to reflect on the project through various lenses: customer issues, technical concerns, project status, as well as personnel, process, and performance issues. The analysis examines actual versus planned results; but even more important, it considers new information. The results of adaptation are fed into a replanning effort to begin the next iteration.

Close

The final phase of the APM framework is closing the project. Projects have a beginning and an end, and each must be recognized. The failure to identify a project's end point can create problems of perception among customers. The end of a project should include cleanup activities and a complete project retrospective to pass lessons learned along to the next project team.

The Essence of APM

In the end, affirmative answers to these two questions form the essence of APM and agile software development: (1) are you delivering innovative products to your customers?; and (2) are you excited about going to work every day? Agilists want to build innovative products — products that test the limits of our abilities — and to create a work environment in which people, as individuals and as teams, can thrive.

Just like Broadway plays always deliver on opening night, APM delivers on time and in accordance with the customer's vision — more reliably than any other approach for high exploration–factor projects. Given the high degree of uncertainty in many new product efforts, given technological change, and given the ebb and flow of staff, reliable results are still obtainable. Given all the maybes, agile methods still deliver — a tribute to the passion, drive, persistence, and ingenuity of project team members.

TRADITIONAL DATA WAREHOUSING

If you have taken part in a data warehousing project, you are aware of the numerous challenges, perils, and pitfalls. Ralph Kimball, Bill Inmon, and other data warehousing pioneers have done an excellent job of developing reusable architectural patterns for data warehouse implementation. Software vendors have done a good job of creating tools and technologies to support the concepts. Nonetheless, data warehousing is just plain hard, and for several reasons:

- Most organizations have not previously built a data warehouse or have only limited experience in doing so.
- Most organizations don't build multiple data warehouses, and therefore development processes don't get a chance to mature.
- Organizations often set out to build an enterprise data warehouse, or at least a broad-reaching data mart, which makes the process more complex.
- Data warehouse consultants have extensive expertise in data warehousing, but not in the organization's business domain.
- Business users typically don't understand what a data warehouse does and doesn't provide.
- Business users often think of data warehousing as a technology-based plug-and-play application.
- As users gain a better understanding of data warehousing, their needs and wishes change.
- Business data always has quality problems that surface in the data warehouse, causing users to question the warehouse rather than the source systems.
- Organizations often view a data warehouse as an IT application rather than a joint venture between business stakeholders and IT developers.
- Data warehousing requires an entirely different skill set than that of typical database administrators (DBAs) and developers.
- Data warehousing requires a multitude of unique skills such as multidimensional modeling; data cleansing; extraction, transformation, and loading (ETL) development; OLAP design; application development; and so forth.

Figure 4 depicts an example of the classic data warehousing architecture, as first described by Kimball [8]. It consists of the following components:

- **Operational source systems.** These are one or more existing systems from which data is extracted, transformed, and loaded into the data warehouse staging database. They are optimized for the daily transactional processing required to run business operations.
- **Staging database.** This database often mirrors the data structures in the source systems and provides a "holding pen" where data can be processed, manipulated, transformed, cleansed, and validated without placing an undue burden on the operational systems.
- **Presentation repository.** Data is extracted from the staging database, transformed, and loaded into appropriate structures for optimized

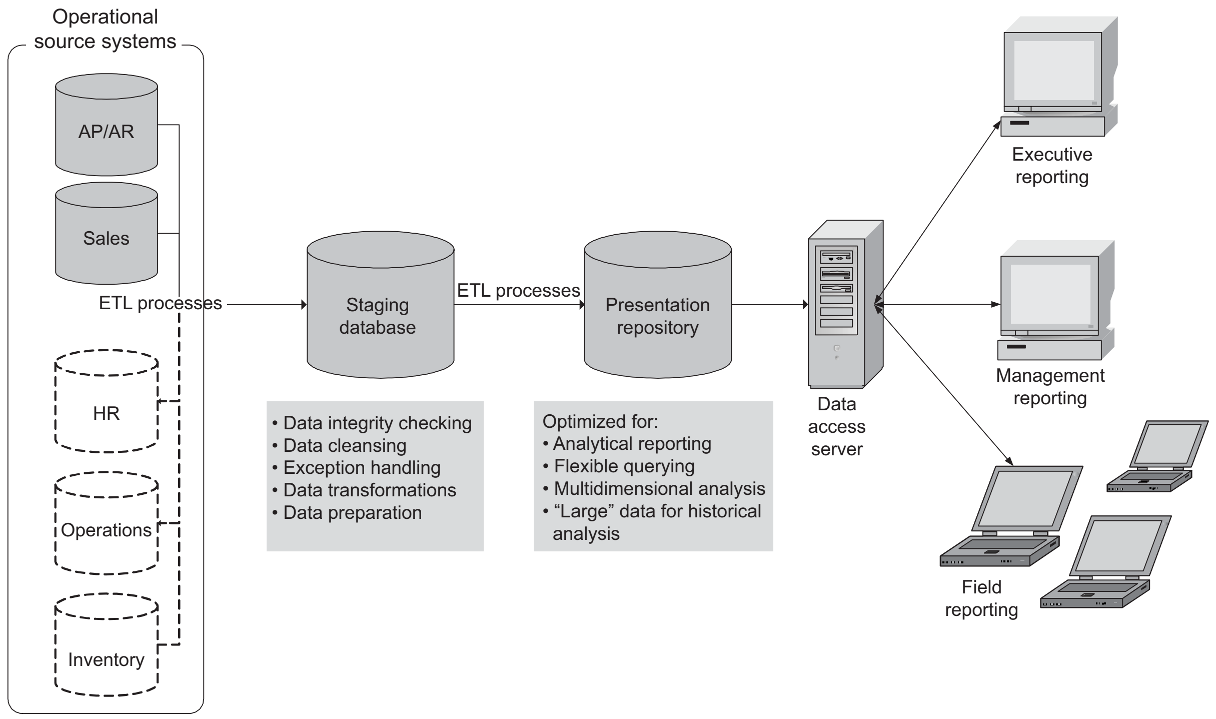

Figure 4 — An example data warehouse architecture.

multidimensional reporting. This system is designed to support the data slicing and dicing that defines the power of a data warehouse.

- **Data access server.** This conceptual server represents the various tools that provide users with access to data, including report writers, ad hoc querying, OLAP, data visualization, data mining, statistical analysis, and so on.

Data Warehousing Technical Skills

Inherent in this architecture are the following aspects of development, each of which requires a unique set of development skills:

- **Data modeling.** This involves design and implementation of data models for both the staging database and presentation repository. Each has unique properties.

- **ETL development.** ETL refers to the *extraction* of data from source systems into staging, the *transformations* necessary to recast source data for analysis, and the *loading* of transformed data into the presentation repository. ETL includes the selection criteria to extract data from source systems, performing any necessary data transformations or derivations needed, data-quality audits, and cleansing.

- **Data cleansing.** Source data is typically imperfect. Furthermore, merging data from multiple sources can inject new data quality issues. As an important aspect of a data warehouse, achieving data hygiene requires specific skills and techniques.

- **OLAP design.** Typically data warehouses support some variety of OLAP techniques (HOLAP, MOLAP, or ROLAP). Each OLAP technique is different and requires special design skills to balance reporting requirements and performance constraints.

- **Application development.** Users commonly require an application interface with the data warehouse that provides an easy-to-use front end combined with comprehensive analytical capabilities and that is tailored to the way the users work. This often requires some customized programming or commercial application customization.

- **Production automation.** Data warehouses are generally designed for periodic automated updates where new and modified data is added into the warehouse so that users can view the most recent data available. These automated update processes must have built-in failover strategies and must ensure data consistency and correctness.
- **General network and DBA skills.** Data warehouse developers must have many of the same skills as those of the typical network administrator and DBA. They must understand the implications of efficiently moving potentially large volumes of data across the network and the issues of effectively storing changing data.

Data warehousing is also hard because of the need for these specialized skills. Most organizations do not have staff members with adequate expertise in these areas.

The Traditional Data Warehousing Process

Data warehousing projects typically follow some variant of the waterfall development approach (see Figure 5). Waterfall and related approaches observe a plan→do model in which exhaustive planning is followed by comprehensive design, development, and testing.

This process is driven by a rigorous up-front requirements analysis with an eye toward collecting and documenting comprehensive user requirements that establish a "contractual" agreement between the developers and users. In this stage, the challenge is to ensure that the users have an accurate understanding of what a data warehouse can and cannot provide and that they have a clear understanding of their own requirements.

Once consensus on requirements is reached, these requirements drive a thorough and detailed data architecture and data modeling effort. This is the core of the data warehouse design cycle, along with other design activities such as volumetric and network load analyses.

By this point in the traditional approach, developers have minimal interaction with users since the parties have already signed off on requirements. Instead, development effort is spent developing formal and detailed data models using tools such as ERwin and Visio. During the design cycle, the design document and data dictionary are typical artifacts that demonstrate progress.

The remainder of the development effort is spent implementing the design; developing ETL code, OLAP cubes, and data warehouse update scripts; and finally, in black-box, white-box, and system-level testing. Final testing may even be handed off to a dedicated QA team to verify that all requirements have been met without introducing new data anomalies.

Finally, when the developers, testers, and DBAs are confident that the data warehouse meets requirements (or, more commonly, when the schedule runs out), users are treated to reviews and user acceptance testing. At this point, the following conditions are most common:

- Users have become more educated about data warehousing.
- User requirements have changed or become more refined.

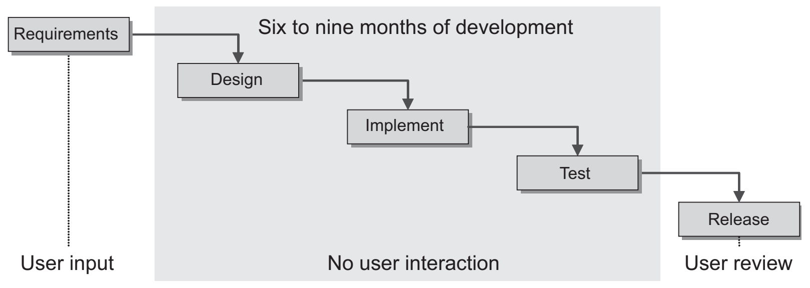

Figure 5 — The typical data warehousing approach.

- Users' memories of early requirements reviews are fuzzy.
- Users have high expectations in terms of anticipating a new and useful tool.
- Developers continue to build the system based on the initial snapshot of user understanding and requirements definition.

All these factors lead to a natural gap between what is built and what is needed.

Data Warehouse Failures

In November 2004, TechTarget polled its subscribers asking, "Have you ever seen a DW project fail?" Among respondents, 52% replied, "More than once"; 10% said, "Once"; 13% said, "Almost, but we made it"; and the remaining 23% said, "Not even close" [10].

A quick Google search on the phrase "data warehouse failure" results in a small library of case studies, postmortems, and assessment articles. While there is no clear definition of what constitutes "failure," Sid Adelman and Cutter Consortium Senior Consultant Larissa Moss classify the following situations as characteristic of project failures [1]:

- The project is over budget.
- The schedule has slipped.
- Some expected functionality was not implemented.
- Users are unhappy.
- Performance is unacceptable.

Simply completing the technical implementation of a data warehouse doesn't constitute success. Primarily end users determine whether a project is successful.

- Availability of the warehouse applications is poor.
- There is no ability to expand.
- The data and/or reports are poor.
- The project is not cost-justified.
- Management does not recognize the benefits of the project.

In other words, simply completing the technical implementation of a data warehouse doesn't constitute success. Take another look at this list. Nearly every situation is "customer" focused; that is, primarily end users determine whether a project is successful.

Assessment

There are literally hundreds of similar evaluations of project failures, and they exhibit a great deal of overlap in terms of root causes: incorrect requirements, weak processes, inability to adapt to changes, project scope mismanagement, unrealistic schedules, inflated expectations, and so forth.

Unfortunately, the traditional development model does little to uncover these deficiencies early in the project. As Jeff DeLuca, one of the founders of Feature Driven Development (FDD), told Highsmith, "We should try to break the back of the project as early as possible to avoid the high cost of change later downstream." In a traditional approach, it is possible for developers to plow ahead in the blind confidence that they are building the right product, only to discover at the end of the project that they were sadly mistaken. This is true even when one uses all the best practices, processes, and methodologies.

What is needed is an approach that promotes early discovery of project peril. Such an approach should place the responsibility of success equally on the users, stakeholders, and developers and should reward a team's ability to adapt to new directions and substantial requirements changes.

ADW FRAMEWORK

ADW is characterized by a highly iterative approach with substantial collaboration between developers, users, and stakeholders. By adapting the Highsmith envision→explore cycle of project management to data warehousing methods, we avoid the disconnect between developers and users as depicted in Figure 5. Figure 6 captures the essence of the envision→explore cycle for data warehousing. It is critical for developers and users to collaborate extensively during the envision cycle and to collaborate frequently and periodically during the explore cycle.

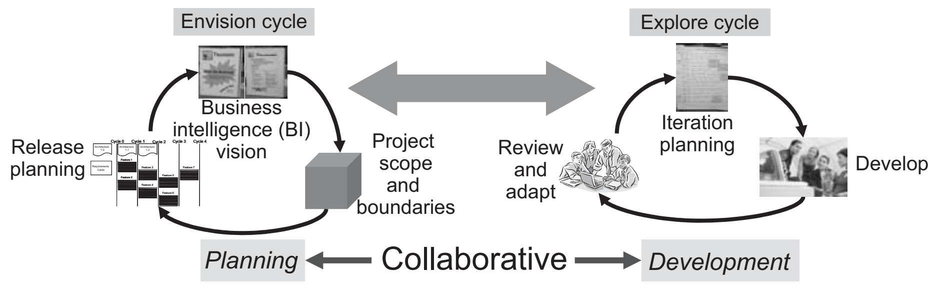

Figure 6 — The agile data warehousing (ADW) collaborative development cycle.

ADW Principles

It's useful to follow a set of guiding principles during data warehouse design and development. When data warehousing poses difficult tradeoffs, these principles often serve as the tiebreaker. Similarly, the AgileAlliance has established a set of principles for software development. The following ADW principles borrow liberally from AgileAlliance principles [3]:

1. Our highest priority is to satisfy the data warehouse user community through early and continuous delivery of working user features.
2. We welcome changing requirements, even late in the course of development. Agile processes harness change for data warehouse users' competitive advantage.
3. We deliver working software frequently, providing users with new data warehouse features every few weeks.
4. Data warehouse users, stakeholders, and developers must share project ownership and work together daily throughout the project.
4. We value the importance of talented and experienced BI experts. We give them the environment and support they need and trust them to get the job done.
5. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
6. A working data warehouse system is the primary measure of progress.
7. We recognize the balance of project scope, timeline, and resources. The data warehousing team must work at a sustainable pace.
8. Continuous attention to the best data warehousing practices enhances agility.
9. The best architectures, requirements, and designs emerge from self-organizing teams.
10. At regular intervals, the team reflects on how to become more effective, then adjusts its behavior accordingly.

Take a minute to reflect on these principles. How many are present in your organization's projects? Do they make sense for your organization? Are they realistic goals for your organization? They are not only commonsense principles but also effective and achievable in real projects. Furthermore, adherence to these principles rather than reliance on a prescriptive and exhaustive process model is very liberating.

ADW Practices

I believe that ADW meshes well with existing data warehousing architectures and practices. I am not advocating ADW as an alternative to the mature practices developed by Kimball, Inmon, Cutter Consortium Senior Consultant Claudia Imhoff, and other data warehousing thought leaders; rather, ADW is largely a framework that redefines the process of implementing those practices.

Some of these practices are directly tied to the APM concepts that Jim discussed previously. Some are adaptations of agile software development practices to data warehousing techniques. Others are based on my experience in trying to tackle the problem of showing users working software early and frequently.

Practice 1: Frequent, Short Iterations

ADW is marked by a highly iterative development cycle whereby each iteration produces a working and demonstrable data warehouse. The working data warehouse instances are architecturally complete even if they are immature relative to the total set of project requirements. Although the length of iterations varies, the aim is to establish an agile rhythm of two-week iteration cycles. This time frame is long enough to build something meaningful and to undo mistakes without it being too costly. But there is nothing wrong with iterations that are as long as 30 days. An iteration that lasts longer than a month, however, runs the risk of the project losing its essential user-developer collaboration.

Presenting users with a working data warehouse early and often accomplishes several goals:

- Users are directly involved in the development cycle.
- Users and stakeholders can see the real progress of development and can appreciate the challenges that developers face.
- Developers can verify that they are on the right track.

After each iteration, a user review takes place; these highly informal meetings involve developers and users. Other than development activities, these meetings should require minimal preparation and planning. The goal of such reviews is to demonstrate the most current working system and to elicit feedback from users. I prefer to follow a customer focus group approach in which the rules are relatively simple. Developers demonstrate, users provide feedback, and a scribe takes notes on user feedback. Developers should not solve problems, troubleshoot, or make commitments based on user feedback. The action taken as a result of feedback is prioritized based on an assessment of the effort required to make changes. Others may observe but should not participate in these meetings.

Periodically, but less frequently than user reviews, executive reviews should be conducted. I prefer to schedule an executive review after every third or fourth iteration, with the goal of demonstrating to executive sponsors and other key stakeholders that the project is progressing and that users' needs are being met. Participants should include the sponsoring executive, stakeholders, and representation from the users and development team. Whenever possible, the executive review should immediately follow the user review. In this case, executives and stakeholders should quietly observe the user review meeting. Then additional project status information can be presented with the active involvement of all parties.

Practice 2: The Conceptual Phase

One exception to the two-week iteration goal is the first iteration, which Highsmith calls iteration zero. This iteration is earmarked for establishing the entire infrastructure necessary for sustainable agile development. Iteration zero includes the following data warehousing activities:

- Capturing user requirements
- Acquiring and configuring the development environment, including hardware, software, and databases
- Identification, acquisition, and understanding of data sources and data quality issues
- Establishing an automated testing framework (which is discussed further later in this report)
- Establishing team roles, responsibilities, and working agreements
- Other activities needed to ensure productive development cycles

Practice 3: Develop Features, Not Functions

Agile development is driven by the completion of customer features rather than functions. The aim is to produce demonstrable features that users can review and evaluate in light of their business requirements.

Like software, many complexities of data warehousing are invisible to users. Typical data warehouse users care about the reports provided by the application, the validity of the values in the reports, and the usefulness of the intelligence in users' daily decision making.

All too often, developers try to "show" users the underlying architectural structures, the ETL logic, and data validation and cleansing code. After all, this is where much of our hard work lies. But despite this hard work, what users truly care about is the reporting application. So the focus of ADW development is on architectural feature spikes: that is, features that are demonstrable to, and thus resonate with, users, such as reports or analytical models. A data warehouse feature is architecturally complete when it pulls data through each of the architectural components and all the functionality at each stage is implemented and tested.

So how do you develop a feature spike in just two weeks? After all, what about all the requisite design, logical and physical data modeling, ETL programming, data cleansing, and other development tasks that are part of data warehousing? My response is twofold:

1. **Define the feature to be small enough to be manageable.** If a single star schema in your warehouse supports several reports, choose a single report. If a single report has a high degree of complexity and underlying logic, identify a meaningful subset of the requirements for that report to implement. Remember, a feature must be architecturally complete, not necessarily mature.

2. **Do the minimum amount of design, modeling, and development required to complete the feature spike.** It is not necessary to fully architect or model the staging database and data warehouse repository to complete the feature. Not only is it acceptable to iteratively evolve the data models and system design, but doing so helps ensure that these artifacts more accurately reflect what you build.

Developers must design and model the feature-related aspects of the system with an eye toward how their design will integrate with the completed system and how it will affect other developers.

Figure 7 conceptually depicts how I think of FDD as it relates to data warehousing. I coach ADW developers to focus on the aspects of each architectural component that are related to the feature they are developing. For example, suppose you are developing an OLAP report for product revenue and profitability. The feature spike might include the following activities:

- Develop the star schema model based on a fact table containing net profit, gross profit, and revenue.
- Identify existing dimensions that can be conformed to the new star schema.
- Identify the source tables necessary to populate your fact and dimension tables.
- Develop the data model for your staging database to capture the requisite source data for this feature.
- Determine the data integrity audits, data cleansing logic, and any data merging requirements for the source data.
- Implement your logical data models for both staging and warehouse repositories.
- Implement the ETL code necessary to source the data into the staging database.
- Fully test the data in the related staging tables according to your QA requirements.
- Implement the ETL code necessary to transform your staging data into your physical star schema tables.
- Fully test the data in your star schema according to your QA requirements.
- Design, build, and populate the OLAP cube.
- Fully test the base table and calculated measures according to your QA requirements.

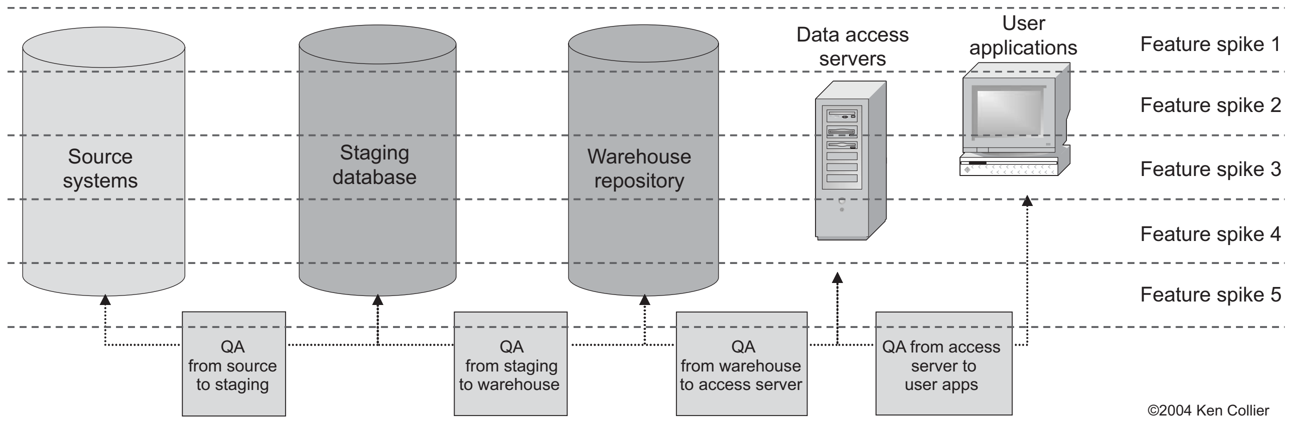

Figure 7 — Architectural feature spikes.

- Develop the report(s) in the user application.
- Fully test the reported values according to your QA requirements.

Note that it is important for agile developers to maintain a global perspective while taking a localized approach. In other words, developers must design and model the feature-related aspects of the system with an eye toward how their design will integrate with the completed system and how it will affect other developers.

This challenge of FDD emphasizes the importance of developer collaboration. There is no substitute for discussing a feature-specific design decision with other developers whose features may be affected by that decision. Nor is this feature-driven approach a substitute for good design. Occasionally a feature may evolve to meet user acceptance criteria but may not be built on the best underlying data models and design. Refactoring is the agile practice of redeveloping an accepted feature using better design principles.

Practice 4: Develop Production-Quality Features

An architectural feature spike refers to a completed feature: that is, a feature that is production-ready and "done-done." This means that all user requirements relative to the feature have been implemented and fully tested. Highsmith likes the guideline that deems features complete when, even if project funding were stopped immediately, users would at least have a working product that meets some requirements. I like the idea that once users review and accept the feature, it requires no further action — it's done!

As additional architectural feature spikes are completed, the system evolves to a fully production-ready state of maturity. In fact, users should already be using completed features to the extent that use does not interfere with ongoing development. As users actively utilize completed features, they are in a better position to provide valuable feedback. Some of this feedback can be immediately implemented with minimal effort, while other feedback may be prioritized for later implementation.

Practice 5: Test-Driven Development

Created by Beck, TDD is "the craft of producing automated tests for production code, and using that process to drive design and programming" [4]. Beck advocates developing tiny bits of functionality based on tiny test cases. The value of TDD is that requirements determine the tests: as soon as your code passes the test, you have met the requirements related to the test.

We can adopt a similar strategy in data warehousing.

Unfortunately it is somewhat more challenging when dealing with potentially large volumes of data. Here is a list of the TDD practices that are proving effective in data warehousing:

1. **Manufacture a test bed.** Contrive a replica of your source databases that contain a handpicked cross section of data in the live systems. The test bed should include significant data volume but should not be unmanageable. The cross section should include records that exemplify all the different types of data found in the source systems, including data that reflects all known quality issues as well as "good" data.

2. **Develop a test-bed catalog.** This refers to a detailed description of the groupings of records found in the test bed. The description should include known data anomalies and other characteristics reflected by the data values and data groupings. It provides a reference for developers to ensure that test cases accommodate all possibilities.

3. **Develop test cases against the test bed.** Since the test bed is a static and contrived database, developers can anticipate the results of each development effort. These anticipated results become the test of success.

4. **Develop tiny and test tiny.** In the spirit of incremental development, small test cases should be developed and then the code should be written incrementally to pass the test. For example, if ETL logic that will replace null revenue values with a 0.00 in the staging database is implemented, the developer can easily and quickly verify whether all nulls are successfully replaced without affecting non-null values.

5. **Test each stage in the architecture.** This is a standard data warehousing QA practice that involves ensuring that the data is correct at each layer in the architecture. By combining this practice with the others listed here, a developer can develop a feature that is of production quality when it is complete.

Practice 6: Automated Testing

Given all the books and articles on data warehousing, surprisingly little has been written about sound testing practices. Data warehouse testing typically involves some degree of integration testing, system testing, data validation, load testing, and user acceptance testing. Unfortunately, practitioners are left to define their own testing methods.

As any experienced developer knows, manual testing is tedious and time-consuming. While traditional developers are in the habit of unit-testing, typical system testing is often handled by a dedicated QA team near the end of the project cycle — a model that is not well suited to the agile practice of testing during development.

As mentioned in the introduction, Ward Cunningham has developed the Framework for Integrated Test, which is an open source framework for automated testing in an agile software development environment [6]. The FIT framework includes code libraries for automated testing using HTML to define test cases and expected results.

While FIT is not directly applicable to data warehouse testing, the principles and approach underlying FIT can be adapted to data warehouse testing with relative ease. Additionally, I am currently working with a development team that automates many of the TDD practices outlined previously.

We have developed scripts that help automate the creation of test cases based on the "source test bed" database. As code is implemented that performs data cleansing, data transformation, and data derivations, the developer can specify an expected result set. A second set of scripts compares the actual result set with the expected result set and produces a report detailing the mismatches found. It is then incumbent on the developer to resolve the mismatches.

We are using this approach for testing each table in the staging database and in the warehouse repository as well as the validation of data in the final reports and models. Coupled with user acceptance testing, this automated approach enables developers to use TDD without the challenges inherent in manual testing.

Practice 7: Incremental Design

Like testing, design and modeling should be integrated into the agile development iterations. While conceptual design largely occurs during the envision cycle or during iteration zero, ADW practitioners should not shirk the responsibility of detailed design. Critics often argue that agile methods short-circuit rigorous and valuable design practices. In fact, agile methods value the importance of good design. Working code, however, is the ultimate test of design.

In one case, I worked on a project in which the company had initiated the project two full years prior to my involvement. The project team spent the first 18 months eliciting user requirements and developing volumes of use-case scenarios to describe all possibilities. It had spent the next six months in system design and was in the process of developing exhaustive design models. By the time the team was ready to begin development, the user community had grown tired of waiting for results and was skeptical that anything would ever be developed.

Ambler has developed Agile Modeling, including a set of principles and practices for effectively applying leading software modeling techniques in an agile development environment [5]. Ambler does not reinvent modeling methodologies. Instead, his practices are based on a set of guiding principles and practices that guide developers to model in small increments and then prove their models with working code, an approach that is inherently agile.

Figure 8 extends the envision→explore cycle by adding a build cycle that describes the development phase of exploration. The build cycle includes model→build →test. During a single ADW feature development effort, the build cycle may be repeated multiple times.

An approach that meshes particularly well with the testing practice of developing tiny and testing tiny is to model a few aspects of the feature and then implement and test just those aspects. Using our product revenue and profit OLAP report as an example, this approach might start by simply replicating source tables in the staging database, building a simple star schema to contain only the basic revenue measure, and populating the OLAP cube with uncleansed data and minimal transformation logic. The modeling activity for staging might simply be the identification of related source tables and their join conditions. The modeling activity for the warehouse repository will simply contain the product revenue fact table and the dimensions that are specified. The next build cycle might add in net and gross profit measures or data cleansing logic.

The agility in this approach is clear. The models evolve as the feature matures, and testing is

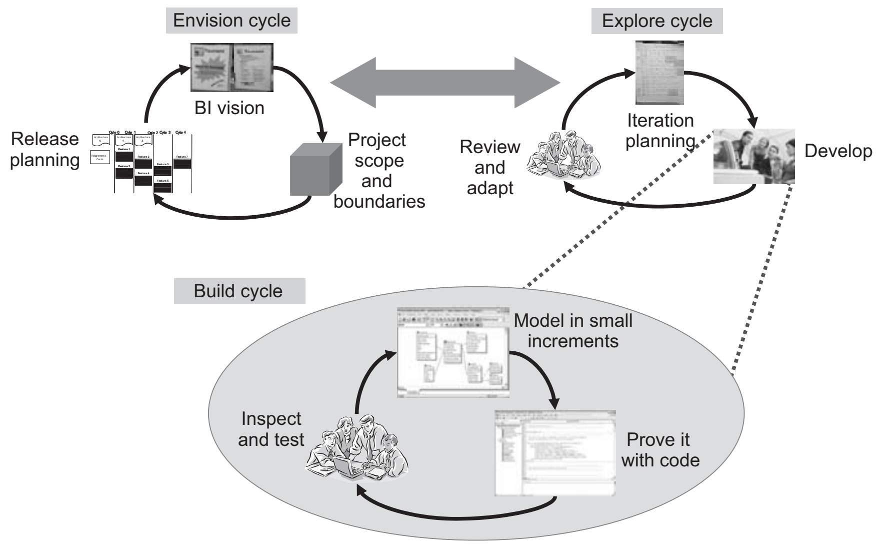

Figure 8 — Modeling in ADW.

tightly integrated into the development cycle. It's important to note that the value of modeling lies primarily in the act of creating the model, not in the artifact itself; and this value increases if the modeling activity is a collaborative and open discussion. I'm sure you can recall various whiteboard sessions with team members where good ideas emerged through the act of discussing a problem and sketching alternative approaches.

Note that modeling is not necessarily equivalent to documentation, although models can be maintained as archival documents for future reference. It is important to understand the cost involved in maintaining your models. Modeling serves one of two purposes: it helps you understand what you are working on or communicate your ideas to others. Once the goal is accomplished, the model should be discarded or be kept and maintained. Too often, we are afraid to discard a model once it has served its purpose. Unfortunately, keeping a model without maintaining it can cause unintended problems. It doesn't take long before the model no longer accurately reflects the working code. Ambler outlines the following principles for modeling:

- **Discard temporary models.** These models have served their purpose.
- **Formalize contract models.** These models are needed to capture agreements.
- **Update only when it hurts.** While it is important to maintain formalized models, this should not be the highest priority.

Practice 8: Barely Sufficient Modeling

This practice is most likely to wake the sleeping giants. For more than a decade, the development mantra has been that success lies in comprehensive requirements analysis and exhaustive design. The inherent problem with this approach is that it is difficult and time-consuming and offers false security, since things inevitably change and initial understandings are often wrong.

There are actually two extremes in data warehouse and database modeling: (1) nonexistent modeling, which leads to patchwork systems; and (2) excessive modeling, which leads to overburdened development and documentation. The aim of Agile Modeling is to find the sweet spot between these extremes that is appropriate for each ADW project (see Figure 9). Highsmith uses the following analogy: If you are trekking through the desert, you will benefit from a map, a hat, good boots, and a canteen of water. But you won't make it if you burden yourself with hundreds of gallons of water, too much gear, and a collection of books about the desert. But it would be foolish to try to make the journey without a minimum of supplies.

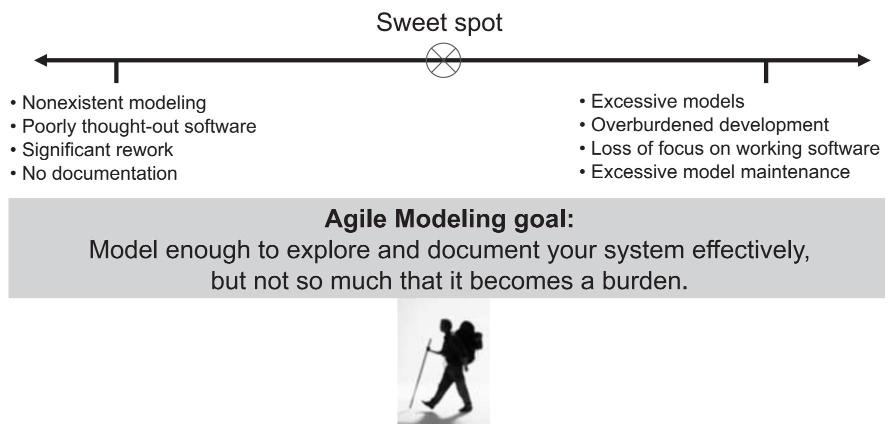
Sweet spot
• Nonexistent modeling
• Poorly thought-out software
• Significant rework
• No documentation
• Excessive models
• Overburdened development
• Loss of focus on working software
• Excessive model maintenance
Agile Modeling goal:
Model enough to explore and document your system effectively, but not so much that it becomes a burden.

Figure 9 — The goal of Agile Modeling.

While modeling is critical to success, effective Agile Modeling requires producing only the models necessary to achieve the intended goal of either understanding or communicating. One of the goals of Agile Modeling is *bare sufficiency*. That is, agile models should accomplish but not exceed their intended goals. Agile models have the following characteristics:

1. **Agile models fulfill their purpose.** If the purpose is to communicate, then the driving modeling questions are "To whom?" and "To communicate what?" If the purpose is understanding, then the drivers are "What is the question?" and "Who needs to be involved?"

2. **Agile models are understandable.** They are developed using the correct "language" for the intended audience, and they provide just enough detail to achieve their purpose.

3. **Agile models are sufficiently accurate.** Models do not need to be 100% accurate. Models must be accurate enough to serve their purpose. If a logical data model refers to the customer as a "client" rather than a "customer," it may still be sufficiently accurate to serve its intended purpose.

4. **Agile models are sufficiently consistent.** Similarly, models need not be 100% consistent. Models must, however, be consistent enough to serve their purpose. If an employee's first name appears as a varchar(30) in the source system model but as a varchar(50) in the staging data model, it is clearly inconsistent. Is this a showstopper? Probably not.

5. **Agile models are sufficiently detailed.** Sufficient detail depends on the audience and purpose. For example, a conceptual architecture like the one in Figure 4 (on page 11) is perfectly fine for a high-level discussion about the flow of data. However, the developers need much more detailed logical and physical data models for the implementation of that architecture.

6. **Agile models provide positive value.** Does the value of the model outweigh the cost of creating and maintaining it? In many cases, flipchart diagrams and digital snapshots are just as valuable as a more detailed ERwin or Visio diagram, and the flipchart diagrams are faster and easier to create.

7. **Agile models are as simple as possible.** Limit the level of detail to only what is needed to serve the purpose. Limit the notational symbols to only what is necessary to serve the purpose. Sometimes a rough block diagram will achieve the same purpose as a more formal entity-relationship diagram with all of its notational power.

Practice 9: Build the Right Project Team

The Agile Manifesto value "Individuals and interactions over processes and tools" is a recognition that no formalized development process can substitute for the right people. We have learned that people are not plug-and-play resources. As Highsmith states, "Getting the right people (which of course implies getting rid of the wrong ones) and the right managers determines project success more than any other factor" [7]. My own experience supports this claim.

Data warehousing requires a broad and varied set of technical skills. Additionally, strong linkage between technical developers and the business needs of users is critical. Finally, executive support for an ADW approach is essential.

Whenever possible, I advocate the ADW project organizational structure that is depicted in Figure 10. In this structure, the overall project has a specific executive sponsor. The BI manager must have a clear understanding of the needs of users as well as an understanding of data warehousing fundamentals to help manage user expectations. The agile project manager must be well versed in the APM methods summarized

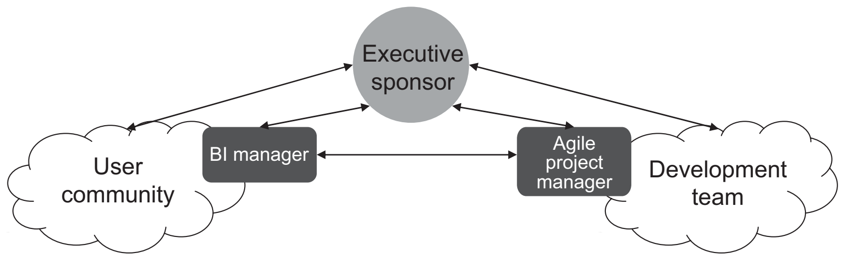

Figure 10 — An ADW organizational structure.

previously. The role of the agile project manager is different from that of a traditional project manager. This group makes up the core project steering committee, which may involve other appropriate participants as well. Project participant roles are as follows:

1. **The executive sponsor:**
   - Provides vision and direction to the agile team
   - Conducts periodic executive reviews
2. **The agile project manager:**
   - Serves as the primary point of contact between the rest of the stakeholder community and the development team
   - Facilitates collaborative design and development
   - Ensures that agile developers have the necessary "tools"
   - Communicates and collaborates with the BI manager
3. **The BI manager:**
   - Represents the user community
   - Liaisons with select users for reviews and feedback
   - Communicates and collaborates with the agile project manager

The development team must comprise the best developers with the appropriate mix of data warehousing skills for the project. Additionally, ADW developers must have the willingness and ability to work in an agile environment. Some extremely talented data warehouse developers are uncomfortable with the lack of comprehensive up-front requirements and exhaustive design that mark traditional development. In my experience, however, once they participate in a few iterations, developers warm to the approach, because it more accurately reflects how developers naturally work.

The selection of ADW users is another critical success factor. It should be apparent by now that the role of the user in ADW is anything but passive. Users must be committed to the project. They must have a sense of ownership of project success or failure. They must also be willing to contribute their time and effort to the envisioning phase and the frequent reviews in the exploration phase. The user community should include a good cross section of the entire user population.

In this report, I periodically referred to "project stakeholders." All projects have some involvement by people who are not necessarily developers or users, but who have some "skin in the game." Author and Cutter Consortium Senior Consultant Rob Thomsett categorizes stakeholders into three levels of participants, each with a different potential impact on the project [11]:

1. **Critical.** These participants can prevent project success before or after implementation. They are showstoppers.
2. **Essential.** These participants can delay your project from achieving success, but you can work around them if necessary.
3. **Nonessential.** These are interested third parties. They have no direct impact on project success, but if they are not included, they can become essential or critical.

WRAP-UP AND FINAL THOUGHTS

This report discusses how to transform agile principles into agile practices that enable development teams to create a higher-quality data warehouse that better accommodates the needs of the user as these needs evolve. By providing users with working features early and frequently, agile data warehouse developers can avoid many of the pitfalls inherent in data warehousing projects. Users' expectations will naturally align with what is being developed and demonstrated. Developers' understanding of user needs will naturally become more refined and more accurate. The system will evolve into a high-value business tool that serves its intended purpose. Responsibility for success will be shared equally among all stakeholders. It is even possible that users may determine that the system is sufficiently complete before the end of the scheduled project cycle, reducing development cost and increasing ROI.

ADW establishes an environment that promotes continuous innovation by satisfying the requirements of users. It also promotes system adaptability by establishing the ability to rapidly respond to change and accommodating future user requirements. ADW can also increase ROI by delivering working applications early as well as reducing development schedules.

In 2002, *IEEE Software* reported that agile methods demonstrated significant improvements in productivity, cost, and cycle time relative to industry benchmarks. The findings indicated an increase in productivity of 15%-23%; a reduction in development cost of 5%-7%; and a reduction in time to market of 25%-50% [9]. Currently, Cutter Consortium metrics experts are conducting a similar assessment of agile methods, and preliminary findings look quite favorable (published results are forthcoming).

As I mentioned at the beginning of this report, I have experienced both data warehousing successes and failures. In my assessment of project failures or struggles, I consistently come to the same conclusions. To be successful, data warehousing projects require a high degree of user/developer interaction, adaptation, and the right people. Thus far, my experiences with applying agility in data warehousing have been a resounding success. Perhaps the best testimonial to the success of ADW is a recent conversation I had with my wife. During the days leading up to the final executive review and project wrap-up, my wife wondered how things were going. When I told her that the project would be finished in two days, she asked, "If this is the end of the project, then why aren't you all stressed out and staying up all night?" She's obviously grown familiar with the traditional project home stretch. She already prefers the new approach.

Agile data warehousing establishes an environment that promotes continuous innovation by satisfying the requirements of users.

As with anything new, the principles and practices that I have been incorporating into ADW are evolving and maturing with experience. I recently started a new ADW project, and so far, so good. However, I fully expect to incorporate the lessons learned during this project into the refinement of ADW practices. So stay tuned, I may have an update for you in my next report. Meanwhile, I hope you will consider running an ADW project in your organization. Try it on a smaller data mart or a revision of your current data warehousing system. If you do it right, I know you'll be pleased with the outcome.

ABOUT THE AUTHORS

Dr. Ken Collier is a Senior Consultant with Cutter Consortium's Business Intelligence and Agile Project Management Practices. He brings more than 15 years' experience in advanced computing and technology to Cutter. With expertise in data warehousing, BI, software engineering, and agile methods, Dr. Collier extends his experience across many industries. Dr. Collier, who is also President of KWC Technologies, Inc., was recently a VP at KSolutions, Inc., where he was responsible for BI solutions and services that included overseeing and managing multiple software development projects. Formerly, he was the head of the BI practice in KPMG Consulting's Knowledge Management Solutions group.

Dr. Collier spent 10 years as a tenured associate professor of Computer Science Engineering at Northern Arizona University, where he developed and taught graduate-level courses in database theory, data mining, and software engineering. There, he also cofounded the Center for Data Insight, a leading center of excellence in data mining and advanced analytics. Dr. Collier holds a Ph.D. in computer science engineering from Arizona State University. He can be reached at kcollier@cutter.com.

Jim Highsmith is a Fellow of the Cutter Business Technology Council, Director of Cutter Consortium's Agile Project Management Practice, and is considered a leader of the agile methodology movement. He is a frequent keynoter at Cutter *Summits* and symposia. Mr. Highsmith is President of Information Architects, Inc. and has 20-plus years' experience as an IT manager, product manager, project manager, consultant, and software developer. He consults with IT and product development organizations and software companies worldwide to help them adapt to the accelerated pace of development in increasingly complex, uncertain environments. Mr. Highsmith is the author of *Agile Project Management: Creating Innovative Products*; *Agile Software Development Ecosystems*; and *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, which won the prestigious Jolt award for product excellence. Mr. Highsmith is coauthor of the Agile Manifesto and a founding member of the AgileAlliance. He can be reached at jhighsmith@cutter.com.

REFERENCES

1. Adelman, Sid, and Larissa Moss. "Data Warehouse Failures." *The Data Administration Newsletter*, Issue 14.0, October 2000.

2. The AgileAlliance. The Agile Manifesto, 2001 (www.agilemanifesto.org).

3. The AgileAlliance. "Principles Behind the Agile Manifesto," 2001 (www.agilemanifesto.org/principles.html).

4. The AgileAlliance. "Test-Driven Development." Version 1.2, June 2003 (www.agilealliance.org/programs/roadmaps/Roadmap/tdd/tdd_index.htm).

5. Ambler, Scott W. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. John Wiley & Sons, 2002.

6. Cunningham, Ward, and Jim Shore. "Framework for Integrated Test." December 2004 (http://fit.c2.com).

7. Highsmith, Jim. *Agile Project Management: Creating Innovative Products*. Addison-Wesley, 2004.

8. Kimball, Ralph. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. 2nd ed. John Wiley & Sons, 2002.

9. Reifer, Donald J. "How Good Are Agile Methods?" *IEEE Software*, Vol. 19, No. 4, July/August 2002.

10. TechTarget. Data Warehouse Failures survey, 2004 (http://searchdatabase.techtarget.com/pollResult/0,294375,sid13_gci921937,00.html).

11. Thomsett, Rob. *Radical Project Management.* Prentice Hall PTR, 2002.

CUTTER CONSORTIUM

Lynn Winterboer Consulting & Training

Agile Success for DW/BI Teams

DW/BI teams that want to benefit from the incremental style of Agile development face a unique set of challenges. Cutter Consortium's Lynn Winterboer will help you succeed.

Introduction to Agile Analytics

This one-day course explains the Agile approach in the context of Data Warehousing and Business Intelligence projects (DW/BI), providing a helpful foundation for the Agile Analytics courses that follow. Participants learn agile concepts and terms and practice applying them to their own DW/BI projects in class. Topics covered include:

- Why Agile?
- The Agile Manifesto: guiding values and principles
- Agile frameworks: Scrum and Kanban
- Agile projects, programs and portfolios
- Requirements: user stories
- Epics and story slicing
- Intro to Agile testing practices
- Intro to the Agile infrastructure
- Getting started: iteration zero

This class is geared to data professionals who are interested in understanding what "Agile" means. Data warehouse and BI project managers; directors and leaders; business users; architects, designers, developers, and administrators; testers; business intelligence practitioners; business analysts and product owners.

Agile Analytics: Program and Project Management

Agility calls for planning for what you know now and being prepared to adjust in the face of change. Agile Analytics planning is a highly collaborative process that includes long-range DW/BI roadmapping to establish future vision, and short-horizon planning of near-term delivery projects. This one- to three-day course will walk you through program roadmapping and project chartering sessions to introduce you to a set of effective practices for facilitating collaboration between technical team members, end users, and management stakeholders. This course will show you how to:

- Charter an Agile BI/DW project using the Agile project management framework
- Coordinate an effective and collaborative program roadmapping session
- Use innovation games for ideation, convergence, and prioritization
- Estimate and prioritize Agile BI/DW projects and features
- Map features, epics, and stories into iterations and releases

Some of the topics you'll discuss may include program roadmapping; program inception; project chartering, sizing, and prioritizing; projects and features; and story mapping.

Agile Product & Project Management

Lynn Winterboer

Consulting, Training & Exec Ed

Cutter Consortium consulting, training, and executive education offerings are developed and presented by its Senior Consultants: you'll benefit from cutting-edge ideas, methods, and strategies presented by the thought leaders who developed them.

Cutter's extensive offerings can be customized to meet your organization's needs and ensure everyone shares the same base knowledge and is well-equipped to take on the challenges of new ways of doing business.

Cutter Consortium
37 Broadway, Suite 1
Arlington, MA 02474-5552, USA
Tel: +1 781 648 8700
Fax: +1 781 648 8707
www.cutter.com
sales@cutter.com

Agile Analytics: Developing in Iterations

While Agile frameworks themselves are fairly simple in concept, putting the concepts into practice takes a mindset shift and daily discipline. This two-day course continues the refinement of the product backlog, expands on estimation techniques, and explains how a DW/BI team steps through each iteration. Participants learn the tactical steps taken by an Agile team in an iteration and the reasons and benefits that drive each step. This class will show you how to:

- Plan and commit to an iteration
- Groom stories in the backlog so they are ready for the Agile team
- Estimate and prioritize user stories
- Deliver value in each iteration
- Build an Agile data team

Some of the topics covered include iteration planning, running an iteration, iteration monitoring, finalizing an iteration, and building an Agile DW/BI team.

Agile Analytics: Design

Agile projects require effective collaborative modeling practices for cross-functional teams. Most projects are more uncertain in the early stages than when near completion. Therefore, the traditional "Big Design Up Front" approach has proven to be costly and risky when much of what was originally designed requires modifications throughout the project.

Agilists focus on "Sufficient Design Up Front," in which they do enough initial design to galvanize developers and testers around a shared understanding of the problem domain, architecture, and data models. This two-day course introduces an Agile modeling approach that strikes the right balance between "Sufficient Design Up Front" and just-in-time modeling.

While this course is centered around technical modeling, it begins with problem domain modeling since just-in-time modeling is very related to the domain of interest. This course will help your teams:

- Identify what is (and is not) an Agile model
- Avoid overbuilding solutions and design only what is needed
- Capitalize on domain modeling
- Identify and understand users, their roles, and personas
- Leverage use cases to develop user stories
- Determine the right level of up-front design
- Deliver immediately with iteration zero
- Minimize unnecessary work
- Define technical debt and understand how it impacts your success
- Use safe techniques for making incremental design changes

Agile Analytics: Testing

Testing discipline, practices and automation are key enablers for an agile DW/BI team. This one-day course provides the testing framework for successful agile data projects, including details on database testing strategies, test data sets, story testing, behavior-driven development (BDD), and test automation practices and tools.

Agile Consulting

Lynn Winterboer provides expert advice drawn from her training and experience. Data warehouse teams and BI teams will benefit from Lynn's deep experience in these fields. She will work alongside your Agile team as she mentors, guides and teaches. Her services include:

- Coaching and mentoring of team members new to Agile
- Assessing and coaching existing Agile DW/BI teams that want to improve their Agile mindset and delivery approach
- Facilitating team problem-solving, working sessions, or planning
- Modeling a particular team role (such as ScrumMaster or product owner) for a time to help identify and train a successor.

Lynn Winterboer will help your team get started on its Agile journey and take Agile to the next level.

Business Intelligence Practice

The strategies and technologies of business intelligence and knowledge management are critical issues enterprises must embrace if they are to remain competitive in the e-business economy. It's more important than ever to make the right strategic decisions the first time.

Cutter Consortium's Business Intelligence Practice helps companies take all their enterprise data, augment it if appropriate, and turn it into a powerful strategic weapon that enables them to make better business decisions. The practice is unique in that it provides clients with the full picture: technology discussions, product reviews, insight into organizational and cultural issues, and strategic advice across the full spectrum of business intelligence. Clients get the background they need to manage technical issues like data cleansing as well as management issues such as how to encourage employees to participate in knowledge sharing and knowledge management initiatives. From tactics that will help transform your company to a culture that accepts and embraces the value of information, to surveys of the tools available to implement business intelligence initiatives, the Business Intelligence Practice helps clients leverage data into revenue-generating information.

Through Cutter's subscription-based service and consulting, mentoring, and training, clients are ensured opinionated analyses of the latest data warehousing, data mining, knowledge management, CRM, and business intelligence strategies and products. You'll discover the benefits of implementing these solutions, as well as the pitfalls companies must consider when embracing these technologies.

Products and Services Available from the Business Intelligence Practice

- The Business Intelligence Advisory Service
- Consulting
- Inhouse Workshops
- Mentoring
- Research Reports

Other Cutter Consortium Practices

Cutter Consortium aligns its products and services into the nine practice areas below. Each of these practices includes a subscription-based periodical service, plus consulting and training services.

- Agile Software Development and Project Management
- Business Intelligence
- Business-IT Strategies
- Business Technology Trends and Impacts
- Enterprise Architecture
- IT Management
- Measurement and Benchmarking Strategies
- Enterprise Risk Management and Governance
- Sourcing and Vendor Relationships

Senior Consultant Team

The Senior Consultants on Cutter's Business Intelligence team are thought leaders in the many disciplines that make up business intelligence. Like all Cutter Consortium Senior Consultants, each has gained a stellar reputation as a trailblazer in his or her field. They have written groundbreaking papers and books, developed methodologies that have been implemented by leading organizations, and continue to study the impact that business intelligence strategies and tactics are having on enterprises worldwide. The team includes:

- Verna Allee
- Stowe Boyd
- Ken Collier
- Clive Finkelstein
- Jonathan Geiger
- David Gleason
- Curt Hall
- Claudia Imhoff
- André LeClerc
- Lisa Loftis
- David Loshin
- David Marco
- Larissa T. Moss
- Ken Orr
- Raymond Pettit
- Ram Reddy
- Thomas C. Redman
- Michael Schmitz
- Karl M. Wiig