Agile EAI

November 2002

Martin Fowler

Gregor Hohpe

Abstract

Enterprise Application Integration (EAI) is a top priority in many enterprises. Requirements for improved customer service or self-service, rapidly changing business environments and support for mergers and acquisitions are major drivers for increased integration between "stovepipe" systems. However, despite increasingly sophisticated EAI suites, enterprise integration remains difficult. Technical, business and political challenges require EAI implementations to be carefully planned, but adaptable to inevitable change.

The apparent dichotomy between careful planning and the ability to absorb changes has been the subject of much discussion in the world of application development. In the past years, a new set of development methods referred to as "Agile Methods" has garnered a lot of attention. Agile Methods are founded on the realization that long-term predictive planning is not appropriate for software development efforts supporting today's business environments. These new methods combine disciplined development methods with the ability to absorb change throughout the development process.

This article examines how the notion of agile software development can benefit integration projects. It examines four best practices proposed by Agile Methods and their applicability in the EAI world.

EAI Challenges

Enterprise integration efforts are as vital to today's enterprises as they are difficult. Enterprise integration by definition has to deal with multiple applications running on multiple platforms in different locations. Current EAI suites offer good solutions to multi-platform, multi-language integration and provide pre-built adapters to most common packaged business applications. However, it turns out that the technical infrastructure presents only a small portion of the EAI complexities. The true challenges of EAI implementations span far across business and technical issues.

Enterprise integration requires a significant shift in corporate politics. Business applications generally focus on a specific functional area, such as Customer Relationship Management (CRM), Billing, Finance, etc. In most instances, the business and IT groups are organized along the same functional areas. Successful enterprise integration requires communication and sharing of responsibilities across units. Application groups no longer control a specific application because the application is now part of an overall flow of integrated applications and services.

Naturally, application integration efforts touch many business and technical aspects within the enterprise. EAI projects require business process modeling at a wider scope than most single application deployments. At the same time, EAI projects also require a significant number of low-level technical developments efforts that generally require more diverse skill sets than application developments efforts. Being able to bridge the gap between facilitating high-level business process decisions and resolving low-level technical issues may be the single most important factor to EAI

success.

A number of new technologies have promised to make enterprise integration easier. One of those technologies is XML. The wide adoption of XML provides a generally accepted data representation layer that allows us to transfer data from one application to another in a platform and language independent manner. However, the frequent claim that XML is the 'Lingua franca" of system integration is somewhat misleading. Standardizing all data exchange to XML can be likened to using a common alphabet, such as the Roman alphabet. A common alphabet can still represent many languages and dialects, which cannot be readily understood by readers. The same is true in enterprise integration. The existence of a common presentation (e.g. XML) does not imply common semantics. The notion of "account" can have many different semantics, connotations, constraints and assumptions in each participating system. Resolving semantic differences between systems proves to be a particularly difficult and time-consuming task because it requires significant business and technical decisions to be made. Also, there are not many tools available to aid with the semantic mapping.

While developing an EAI solution is a difficult task in itself, operating and maintaining such a solution may be even more daunting. The mix of technologies and the distributed nature of EAI solutions make deployment, monitoring, and trouble-shooting a complex task that requires a combination of skill sets. In many cases, these skill sets do not exist within IT operations or are spread across many different individuals. Anyone who has been through an EAI deployment can attest to the fact that EAI solutions are a critical component of today's enterprise strategies, but make IT life harder, not easier.

Agile Methods

Traditionally, software engineering efforts have responded to complexity and uncertainty with an exhaustive up-front planning phase. These predictive methods were based on the presumption that rigorous up-front planning is ultimately able to eliminate execution uncertainty. As a result, many projects dove into "analysis paralysis" and created huge requirements and design specifications, just to find that during the project execution changing requirements and new findings threatened to impact the carefully crafted project plan. In many occasions, the result was a cancelled project, a delayed delivery or delivery of a system that did not meet the business needs. Many project managers learned the hard way that "Users do not know what they want until you give them what they asked for."

Recently, a new family of development methods, referred to as "Agile Methods", took a new approach to project management. Recognizing the inability to eliminate change and uncertainty in a business environment, Agile Methods adjust the development process such that it is able to absorb change and uncertainty. The underlying values of agile development methods are represented in the following "Agile Manifesto" [1]:

Agile Methods value:

Individuals and Interactions	over	Process and Tools
Working Software	over	Comprehensive Documents
Customer Collaboration	over	Contract Negotiation
Responding to Change	over	Following a Plan

This means, while Agile Methods value the items on the right, they value the items on the left more. For example, while design documentation is important, when faced with an either-or decision, Agile Methods value working software over a stack of papers. Agile Methods are certainly not suited for every type of project. Pacemakers and missile control systems do tend to have the benefit of stable requirements and are best served by a rigorous up-front planning and design effort. On the other hand, most business applications are driven by market forces and demands and are therefore subject to constant change. For these projects, Agile Methods provides a more flexible process, which can absorb change at a higher rate than traditional methods.

Agile EAI

Agile Methods have demonstrated impressive success stories in the world of enterprise application development[2]. Enterprise integration projects are characterized by a similar or even greater level of complexity and uncertainty than enterprise applications. Additionally, integration projects increasingly contain significant application or business logic. So how can some of the benefits of Agile Methods be brought to bear on integration projects?

Agile Methods consist of a collection of best practices that have been proven to support the need to absorb change while delivering a high-quality solution. Many of the practices can be applied in the world of application integration with small modifications:

Iterative Development

Many projects end in the proverbial death march. Most of the time, this crunch near the end of the project occurs because during the final testing and deployment, all the bad surprises come to daylight: the piece of code that was 90% complete a few weeks ago is still not done. The users realize that a feature works differently than they had expected and the module that was slipped in at the last minute caused defects in other parts of the program. As much as we hate this final part of the project, it does help uncover many problems and provides us with an accurate read on the health of the project. If we could go through this final phase more frequently, chances are that it would be much less painful and would give us a more accurate read on the actual project progress. Developing and deploying a solution in multiple iterations allows us to do exactly that.

Developing in multiple, shorter iterations realizes a number of additional benefits. Releasing a limited set of functionality early allows users to see a part of the total solution and to provide feedback throughout the development cycle. This significantly reduces the risk of missing critical features or delivering features that are no longer required. It may also allow business users to start realizing business benefit sooner. In addition, iterative development lets

developers and architects familiarize themselves with the specifics of the systems that are to be integrated within the context of a small piece of functionality. We can think of it as the "Hello World" of EAI.

The benefits of releasing functionality in multiple iterations applies to enterprise integration just as well as it does to application development. The key is to identify meaningful subsets of functionality that can be reviewed by the users. Similar to application development, there are a number of design decisions that have to be made at the beginning and cannot be significantly changed throughout the iterations. This includes fundamental architectural decisions such as naming standards and the underlying network architecture. These fundamentals are usually defined in a so-called discovery phase that precedes the first iteration.

Simple Design / Evolve

The concept of simplicity goes hand-in-hand with iterative development. In order to deliver value in a shorter amount of time, the solution has to be simple in nature. Rather than developing a huge framework over six months, we solve a specific problem in a few weeks and demonstrate that we understand what the business needs.

Simple does not mean unsophisticated or sloppy. Simple means that we attack the problem straight on and do not create an enormous infrastructure to support a minuscule portion of functionality. Building infrastructures is fun. It lets the developers work in an idealized world of elegant concepts and generic solutions to theoretical problems. We have seen many very, very beautiful infrastructures – works of art! Well, that was usually at the end of the "infrastructure phase" before the ugly reality of vague and changing business requirements marred the beautiful picture. What a shame. By growing the infrastructure as the functionality grows, we can ensure that the infrastructure supports what is really needed by the business. In the end, an infrastructure that was properly evolved leads to a better result than a beautiful ivory-tower infrastructure that was subsequently bastardized to meet real requirements.

How can an infrastructure be evolved without deteriorating into a complete mess? Two key factors: discipline and frequent testing. Evolving the infrastructure and functionality does not imply hacking away. To the contrary, a more disciplined approach is needed – especially in the world of EAI that is ripe with loosely coupled, cross-language, cross-platform interactions. Before we start developing, crucial standards have to be established: subject / queue / channel naming, component naming, documentation standards. As the system evolves, these standards and guidelines will allow developers to get a quick and accurate read of the system at any point in time and allows them to make changes that are in line with the overall solution philosophy.

Automated Testing / Integration

The second key factor to allow evolution is frequent testing. In order to keep the whole development team from testing eight hours a day, these tests better be automated. Running a complete test suite should be as simple as clicking a button and reviewing the report later on. In the world of custom application development, the suite of public domain 'xUnit' frameworks have become the tools of choice for automated unit testing. These frameworks execute all test

cases and compare actual results with desired results. Any deviations are reported. If all test cases succeed, we see the infamous 'green bar' (the progress bar turns red if a test case fails)[3].

It turns out that effective unit testing is more difficult in EAI development than in application development for a number of reasons:

- The minimal unit of test in the EAI world tends to be larger and more data driven than in application development. Testing a single method of a Java class generally requires a moderate amount of preparation. On the contrary, testing a single integration function may require complex data setup inside multiple packaged applications.
- Most application testing is synchronous. A specific function is called and the results are compared to the expected result. EAI functions are often times asynchronous. An EAI test case may produce results in an asynchronous manner. Results may arrive later or not at all.
- EAI environments are inherently heterogeneous. In an application development environment, a Java-based tool such as jUnit can cover the vast majority of unit test cases. The EAI solution may span across multiple languages and platforms, including mainframe legacy systems. This makes the capture and comparison of test results much more difficult.
- Testing tools are not yet readily available. Many EAI vendors focused more heavily on integration functionality than testing support. In many cases, testing tools have to be created and are vendor specific. Some integrators (such as ThoughtWorks) have created proprietary testing tools. When you select an integrator to assist with EAI implementations, ask them about their approach to testing and specific tools they can bring to bear.

Close Customer Collaboration

Continuous customer collaboration is another critical enabler to iterative development and short feedback cycles. This successful practice is arguably easier to perform in front-end application development than in enterprise integration, because infrastructure software is inherently difficult to demonstrate. Many of us may have seen an EAI demo where some person clicks a button on one machine and the screen changes on the machine next to it. While interesting to watch, this demonstration may not move us to spend a six or seven digit amount on an enterprise EAI initiative.

On the other hand, the need to demonstrate business features to a customer frequently reminds the integration teams of the ultimate deliverable -- business value. As such, it is a critical metric for the EAI development teams to come out of the dark hiding places of infrastructure retrofits and measure their success by the value that was provided to the business.

Conclusions

Enterprise integration is a critical part of today's enterprise strategies. Integration projects are equally or even more difficult than most application development efforts. This difficulty stems from complexities in both the business and technical domains. Given the spotty track record of large application development efforts, embarking on a complex

integration project may scare many business sponsors and IT departments. Agile development methods have offered the development communities some relief and demonstrated critical project successes. With some modification, many of the principles and techniques used in agile software development can be applied to the world of enterprise integration and help make the next integration project a successful one.

Authors

Martin Fowler is the Chief Scientist at ThoughtWorks, Inc, an Internet systems delivery and consulting company. Contact him at fowler@acm.org.

Gregor Hohpe leads the Enterprise Integration Services practice at ThoughtWorks, Inc., a provider of application development and enterprise integration services. For the past four years, he has been helping clients around the world design and implement enterprise integration solutions. He can be contacted at ghohpe@thoughtworks.com.

References

[1] http://www.agilealliance.com

[2] Caterpillar digs into Agile development methods: http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,67016,00.html

[3] http://www.junit.org

