Agile ERP: "You don't know what you've got 'till it's gone!"

Gerard Meszaros
clearStream Consulting
gerard.meszaros@acm.org

Janice Aston
Canadian Pacific
Janice_Aston@cpr.ca

Abstract

ERP systems are known for their highly integrated nature and proprietary development tools and processes. There are a number of things we just take for granted in .Net & Java that turn out to be critical for doing Agile. This paper describes the impact these implicit assumptions had on applying agile development processes on a medium-sized custom development project based on SAP NetWeaver 2004s. We describe the problems encountered and how the project team overcame them.

1. Introduction

1.1 Company Background

Based in Calgary, Alberta, Canadian Pacific (CP) is a Class 1 North American railway providing freight transportation services over a 14,000-mile network that extends from the Port of Vancouver in Canada's west to the Port of Montreal in Canada's east, and to the U.S. industrial centers of Chicago, Newark, Philadelphia, Washington, New York City and Buffalo.

Canadian Pacific was founded in 1881 to link Canada's populated centers with the vast potential of its relatively unpopulated west. This incredible engineering feat was completed on Nov.7, 1885 - six years ahead of schedule - when the last spike was driven at Craigellachie, B.C.

Consistent with over 120 years of traditional engineering project experience, most Information Technology projects at CP embrace traditional waterfall or document-driven development methods. The stated preferences are to "Reuse, Buy, Build" in that order. In an attempt to improve its IT delivery capabilities, CP has recently started to experiment with agile methods on the .Net "Build" side of the house. The PR3 project was the first SAP project to attempt applying full-blown agile development at CP.

1.2 Project Background

The Price Right 3 (PR3) project was chartered to replace a failing application (known as Price Right 2 or PR2) and align distinct business processes (applications). Despite the rich functionality in the existing application, the "hopping" between multiple applications to complete the Lead to Contract business process (See Figure 1.) was highly unproductive. The primary driver for our business case was increased usability resulting in productivity gains thereby freeing up account manager time for proactive selling to increase revenues. In keeping with an agile mindset, this drove many of our decisions.
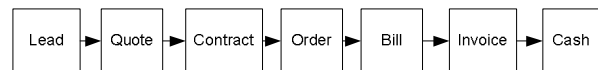
Figure 1. The Lead to Cash business process

The PR3 project was part of a multi year pricing program designed to deliver a custom built pricing solution. The first project, Price Right Interline (PRI), provided the ability to exchange prices with other railways using web services. That project was executed in .Net using agile development and project management processes augmented by key UxD processes [1]. It delivered on time, under budget and with exceptional quality and was considered a resounding success by the business, not just IT. Riding on this high we felt confident tackling the next phase, replacement of the legacy quote application. Wrong!

1.3 Why Custom Development in SAP?

Part way through the planning of the PR3 project, CP signed a strategic agreement with SAP Canada and adopted an "SAP First" philosophy. After saying goodbye to a high performing .Net development team, the remnants of the team soldiered on. We ramped up

the team with "agile friendly" SAP resources. A motley crew of Agile purists, .Net bigots and Agile newbies set out on an adventure and had no idea what we were getting into. We started by comparing the requirements for PR3, which were by that time quite well known, with the capabilities of the SD module of SAP's ERP offering and determined that there was a sizable gap both in functionality and in usability. The business made it very clear that they would not accept anything that was any less usable than the existing PR2 and PRI applications.

On the plus side, the SAP SD module implements the Quote to Bill process and is pre-integrated with the Finance module that implements Bill to Invoice to Cash (See figure 2). SAP also comes with a lot of standard functionality that can be accessed from within both standard SAP code and from custom-written code. To leverage these capabilities and the pre-integrated functionality we decided to develop a custom-built quote management front end in ABAP (SAP's programming language) to build the quotes that would later flow through the Quote-Cash process. (We considered doing the front end in .Net but decided that it would involve too much duplicated effort between the .Net code and the ABAP in the back end.)
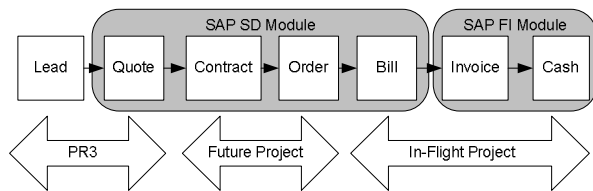

Figure 2. The Lead to Cash process in SAP at CP

1.4 Why Agile?

For both the PR3 business lead and the PR3 project manager, using Agile on the PR3 project seemed like a natural fit. Both had been through the PRI project which had delivered excellent results. When asked for his reaction to the agile process, the PRI project business lead had responded to the effect that "There was no way we (the business) could have described this application in requirements documents" and that agile was the only way that made sense. We also had several experienced .Net agilists who had been through the PRI project to act as business analysts, UI designers, Fit test [3] automaters, agile process coaches as well as retrofitting the PRI application to use the SAP pricing engine as its core.

From a technical perspective, using agile with SAP also seemed to make sense because SAP is delivered as a fully working, though standardized and standalone system. This means one can start the project with working software and evolve the system from there by configuring various aspects of its behavior and hooking up various integration interfaces with other applications in the enterprise. Further supporting agile development, SAP comes with its own Recorded Test tool, eCATT and xUnit test framework [2] ABAP Unit. So we said to ourselves "ABAP is just another object-oriented programming language; how hard could it be to use agile?!"

The rest of this paper describes all the things we take for granted in the .Net, Java and scripting language worlds that make agile development possible and how we found them to be lacking in the SAP ecosystem.

2. Product-Centric Viewpoint

SAP is in the business of selling business process automation. The applications they sell are each centered around a particular business process which they automate from end to end.

SAP benchmarks the business process before they automate it and they market the automated version of the process as an industry best practice. The SAP product specialists one hires to help one implement SAP know the product inside and out and will help you determine which "switches" to flip to get it to behave the way you want it to – assuming that what you want is one of the variations that SAP has allowed for. It is widely felt in the SAP community that most companies should change their business processes to match what SAP has implemented. Unfortunately, when your business process is your competitive advantage, reconciling these opposing views can be a challenge.

This "the product works this way" mindset is at serious odds with the "whatever the customer wants" mindset of an agile team. We had many discussions where the SAP side of the team told the business side of the team that "we cannot do it the way you want because SAP does it this (other) way". Eventually, though, we often came to the realization that we had to do it the way the business requested. Fortunately, SAP often provides a mechanism called "user exits" to override the algorithms it implements. For example, unit of measure conversions between "per car", "per net ton" and "per hundredweight" were "just math" in .Net but in SAP we had to configure the SD module to do this conversion. In the end, however, we could not get SAP's rounding of the price to work properly so we had to override the unit of measure conversion via SAP-provided "user exits".

3.   Specialization of Roles

The SAP architecture is highly configurable. To achieve the flexibility required to accommodate the needs of SAP's breadth of clientele, almost everything is data-driven. There are four kinds of data in an SAP system: The most volatile data is the Transactional Data such as Quotations, Billing Documents and Invoices. Master Data is the more stable data that Transactional data references; things like Customers and Materials. Configuration Data is used to control the behavior of the SAP-provided functionality; it may determine which variation of a process is executed or it may define installation-specific algorithms or business rules. Even the executable ABAP code itself is considered to be data that is stored in tables and then compiled into memory as needed.

The specialists who configure the product are called "functional analysts". They need such deep knowledge therefore most SAP resources are highly specialized and know only one or two modules. SAP further encourages this by offering extensive certification courses without which it is hard to get hired on by potential employers.

Even the ABAP programmers are very specialized because of the need for intimate knowledge of the SAP module's table structures and programming APIs. When we started recruiting for the SAP incarnation of the PR3 project, we quickly encountered this issue and had to go back to the recruiting well many times before we found resources with the right module specializations and a willingness to try agile development at our expense. We ended up hiring two SD functional analysts, and four ABAP developers each of whom had SD experience. Some of the ABAP developers had an additional area of expertise. Contrast this with the .Net world where "a developer is a developer" and can be expected to do most anything.

Many of the task a .Net developer would do for themselves, such as installing software, configuring software components, etc. cannot be done by normal ABAP developers in SAP. A special skill set known as "Basis" needs to be asked to do these things. It turns out you need these Basis folks to do all sorts of stuff for you at pretty regular intervals throughout the project and we really should have hired one for our project.

Most traditional IT projects at CP have a Solution Architect who is responsible for the integrity of the application and how it interacts with all the other systems. This role doesn't seem to exist in the SAP world and the few people who could do it are very senior and very much in demand. On most simple implementation projects, the functional analysts work with the business to map out the business processes,

determine how to configure the product to automate the process, and write up design documents telling the ABAP developers what custom code they need to write. (See Fig. 3.) Based on this we asked one of our functional analysts to carry out the role of solution architect. He agreed reluctantly when we promised him lots of support and coaching. But after several months it because very apparent that architecture really isn't part of the functional analyst's job description which is very much a product-centric role. We ended up having an experienced Java/.Net solution architect play the role instead.
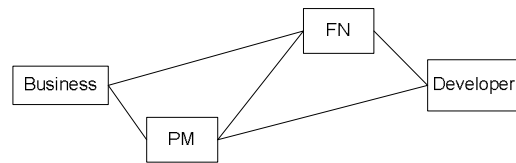
Figure 3.   Traditional SAP project roles and communication paths

The extreme specialization of roles made estimation and velocity tracking a bit of a challenge. Traditional agile methodology assumes that people are generalists who can play whatever roles are needed at a particular time. This is certainly not the case with functional analysts (who only write documents and configure the product) and ABAP developers (who only write code.) We started off by assigning separate story point counts for functional work and for development work and keeping track of team velocity for each. After a while it became obvious that development was the bottleneck so the importance of configuration points diminished; we still noted them while estimating but we didn't bother calculating "configuration velocity".
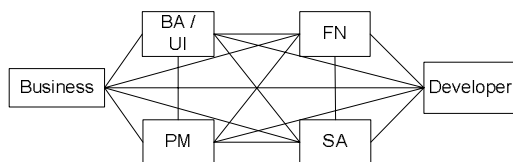
4.   Communication & Collaboration

Agile development processes put a lot of emphasis on collaboration and rich person-to-person communication. This proved to be a challenge due to several factors in play in the SAP community at CP.

The first challenge is related to the specialization of roles: traditional SAP development is a document driven process. The functional analysts examine the business processes of the company and determine how the product needs to be configured to automate those business processes. The processes are captured in one set of documents and the planned changes to configuration data in another. If any code or tables are required, these are documented and passed on to the ABAP developers for development. The ABAPers often have to ask the Basis group to do the really technical infrastructure things for them, things like

license keys to activate various bits of functionality, security, investigating server performance issues, etc..

On their previous projects, our ABAP developers rarely spoke to business people and as such had not developed the ability to communicate using the language of the business. We had many lengthy discussions where it turned out that everyone was in violent agreement about something but the SAP-specific language being used by the SAP people was obscuring this fact. Further complicating things is that SAP AG's mother tongue is German so much of the code and data schema uses German terms or abbreviations. As the more technical non-SAP people on the project learned new SAP terms, they would user their SAP "decoder rings" to provide simultaneous translation to plain English for the business.

Figure 4. PR3 project roles & communication paths

We suspect that the document-driven mindset is also the root cause of our difficulties getting people to collaborate, especially cross-functionally. We had to push hard to get developers to pair program and for developers and functional analysts to sit down side-by-side to work on stories that required both configuration and coding. We still see occasions where they fall back to doing their own kind of work and then "throwing it over the wall" to the other specialty to do their job and this often results in several trips back-and-forth over the wall before the functionality actually works.

Another challenge has been instilling a mindset of incremental delivery. This has been especially difficult with the functional analysts who need to do a lot of up front analysis for database extensions and data loading programs. We defined user stories for this analysis but work tended to go on for iteration after iteration. We also saw the same problem with some of the ABAP developers working the data load programs; we asked for an initial delivery of "quick & dirty" data load to get our development environment up and running. What we got in the same timeframe was a half-built (not working) production quality data load program!

Another standard practice on agile projects is the use of a self-directed (not self-managed) team. We have large information radiators to convey what user stories and tasks need to be done next and the team has gotten pretty good at working on the next most important item. Where the team has had a few challenges is on work processes improvements being initiated by a peer. The process changes we have

implemented were usually initiated by the project manager and not by individual team members. We also have had to poke fairly hard during our biweekly iteration retrospectives to get suggestions for ways to improve the process.

There has been some reluctance to embrace more collaboration development practices such as pair programming and we have had occasions when a developer has "gone dark" (reverted back to working on their own) for several days at a time. In this instance the project manager initiated discussions with the development team resulting in a mini-QA process consisting of mini-design discussions and frequent code inspections if pairing wasn't being used.

5. Server-Based Development

In .Net and Java our developers are used to working on their own workstations and choosing when to commit their changes to the continuous integration server. This allows everyone to work independently and expect all tests to run green when their changes are done.

One of the biggest and highest impact surprises we encountered early in the project was that ABAP development in SAP is server based. Developers run the SAP GUI on their own PCs but all the ABAP code is stored and executed on the development server. This means that every time a developer saves and compiles their code, all the other users on the server see the change immediately. This meant that we would not be able to run all the unit tests and expect to get all of them to pass because someone would always have some functionality in progress which results in code that didn't compile or didn't pass its tests.

ABAP development teams traditionally get around this by using a pessimistic locking model; this involves dividing up the work based on the affected objects or function modules so only one developer needs to touch each one. This makes story-driven development difficult because stories rarely are confined to just one object. It also makes running regression tests, whether unit or functional, all but impossible as we quickly found out when a test tool vendor came to show us their wares and co-habitation of the development environment quickly caused disruption on both sides.

5.1 The "Agile Zone" to the Rescue

To address the server-based development issues, we worked with some senior people at SAP Canada to come up with the concept of an Agile Zone consisting of one dedicated developer workspace (on a shared server) for each pair of developers and one for our business testers. We initially proposed installing SAP

on each of our PCs and they countered with a proposal to build a proper SAP development server for us and then to clone it n times on the same physical server. At first, they were sceptical about how anything other than the traditional pessimistic locking model could work for server-based development. We had to educate them about how the agile update-edit-test-checkin process works.

Because this was considered a rather radical idea within the CP SAP community, we spent a great deal of time gaining support and buy in for the concept. We had to run this idea past all sorts of committees within CP's IT organization to get their blessing even though we were prepared to pay for it from our own project funds. In the end we were allowed to go ahead and build it. This lead to our next roadblock: a 12 week lead time for the server from our IT Infrastructure outsourcer.

We convinced the infrastructure supplier to install an existing commodity server (only 4 weeks lead time) so that we could start having SAP installed and configured for us. We ordered a memory upgrade which would be installed whenever it arrived.

We worked with our Basis expert who came up with the Transport strategy (English translation: Intersystem Code promotion policies). He also finalized the configuration of our system and it's clones. When we turned everything on, all four SAP instances on the server ground nearly to a halt. The problem was eventually traced to a faulty SAN configuration for our page files. The ABAP developers soon became experts at implementing the Transport-Red-Green-Transport pattern (English translation: Update, Write Tests, Write Code, Check-in"). This allowed us to organize the work around user stories rather than the objects they modify. It also allowed us to write and run ABAP Unit tests and expect a green test run before transporting our changes to the integration Pre-Dev server.

At one point we had to shut down all our developer workspaces and have everyone work in the main Pre-Dev server while a system upgrade was done. You should have heard the developers complain about having to go back to how they had been so content to work for so many years!

6. SAP is an Integrated System

We initially defined a release plan that delivered working software in 5 releases over a year and a half. Based on our experiences in the .Net world where we could deploy our application on our own schedule, we simply assumed that we would be able to fly under the radar and deploy whenever we wanted. Unfortunately, as soon as we talk about putting code into the ERP Dev environment, strict governance processes kick in to review changes and potential impacts.. Welcome to the "Integrated System"! Discussions to identify the "integration points" (areas of software overlap) early and develop ways to "pre test" the integration without all our software being present merely opened the "integrated system" can of worms even further.

One of the "benefits" of SAP is that it is a fully integrated system that implements an entire business process – in our case, Lead to Cash. This requires that the entire business process be thoroughly understood from end to end to avoid "rework". In our case, we were only implementing the Lead to Quote part of the process (See Figure 2). The Order to Invoice to Cash process was already implemented by another project and the Quote to Contract to Order parts of the process are planned for sometime in the future.

Our agile philosophy was to "not close any doors" and that some rework was inevitable, acceptable and manageable. The established SAP sustainment group at the company thought otherwise; they wanted to avoid any rework at all costs. This led to bitter disagreements in design reviews and to protracted speculation and negotiation about how the functionality of the two projects, while never actually interacting, would be aligned. So much for flying under the radar!

A further issue is that we must align our schedule with the implementation of the latest service packs from SAP because they add important usability-related capabilities to the relatively new ABAP Web Dynpro UI framework we are using.

The "Agile Zone" has helped mitigate many of the impacts of the "integrated system" during the development phase of the project by giving us complete control over our development environment. We can do our development without being impeded by the waterfall QA processes of the sustainment group and verify that our design works before we show it to them. The discussion thus changes from "could this possibly work" to "does this line up with how the company wants to do things". We expect some rework at this point but our safety net of automated tests should let us make whatever changes we need to make quickly and safely.

The sad reality is that deploying software is well beyond the control of our own project. Initially, no one could even tell us how long this would take because the organization was still trying to figure out how to manage multiple simultaneous projects in the ERP space. We are now estimating a 3 month delay between when we have fully tested, ready to deploy software in our "Agile Zone" and when it will be deployed to PROD.

7.   What's Next?

As of early May, 2007 we are on our 16th 2-week iteration. Every iteration starts with an iteration planning meeting and ends with a big-screen demo and an iteration retrospective. The business is thrilled with the way the quote management application is taking shape; they believe it will be even easier to use than the fat client desktop application they are using today. (Who would have thunk they would find an SAP application to be more usable!)

We are preparing manual story tests for each user story and writing unit tests in ABAP Unit. We are using Fit.Net via the SAP .Net Connector to prepare component tests for the pricing configuration. We are reusing the Fit functional tests we prepared for the Interline Pricing (PRI) project as regression tests for the hybrid SAP/.Net product. We are also automating some functional "smoke tests" for the web-based user interface.

Our team is starting to mature in its adoption of Agile practices. Some team members can no longer imagine implementing an SAP solution in a traditional fashion. They claim there is no going back and we have spoiled them for their next project.   To address the cultural clashes we have segregated all integration activities managing them in a more traditional fashion. We are agile at the core with a traditional wrapper facing outwards.

We have recently participated in the required architecture reviews, design reviews, code & configuration walkthroughs with the sustainment group. There is a growing realization that we are not the "undisciplined cowboys" as originally thought. In fact they are rather impressed with some of our development practices. Who knows, maybe they will consider adding them to their development best practices.

8.   Conclusions

Be prepared to challenge many assumptions that you would take for granted in the .Net/Java worlds. Both the nature of the server-based environment and the associated BDUF/waterfall mentality it engenders in the established ERP development community can be serious impediments to implementing full-on agile practices such as test-driven development. Some agile-friendly practices such as Pair Programming are not affected but the much-too-continuous nature of server-based development makes automated testing and storytest-driven development challenging. With some effort and persistence, we have been able to build an Agile Zone development environment that resembles what Java and .Net developers take for granted.

Many of the biggest obstacles are cultural. The "You should change your business to match SAP's best practices" mindset is definitely at odds with the Agile business value paradigm.   The extremely traditional document-driven development process is probably the least agile-friendly aspect of ERP systems but it is certainly not unique to ERP. Some BDUF practices are necessary in the "integrated environment" of an ERP system, most notably, alignment of key business data such as customer but we are doing emergent design of the user interface software and the business logic behind it within our two week development iterations.

This Agile ERP road has been long and full of potholes. Would we run another Agile ERP custom development project again? It depends on the constraints. Compared with our initial projections of the .Net version of the project, our costs have more than doubled. We attribute this to the higher cost of SAP resources compared to .Net resources and the reduced productivity caused by the less-than-state-of-the-art development tools. At this time we are not leveraging the full power of SAP's integrated system because only one step of the Lead to Cash process is in scope. Therefore, assuming there is significant custom development involved, the question to ask oneself is whether the integrated nature of SAP provides enough value to overcome the additional development cost.

Assuming one has decided to do the custom development in ABAP, is it worth doing it in Agile? We would answer with a resounding YES! Implementing agile development of an application in a server-based ERP system definitely has some challenges but it can be done and the benefits of increased business involvement, buy-in and satisfaction are huge.

9.   References

[1] Meszaros, Gerard, Janice Aston, Adding Usability Testing to an Agile Project

[2] Meszaros, Gerard, *xUnit Test Patterns – Refactoring Test Code,* Addison Wesley 2007

[3] Mugridge, Rick, Ward Cunningham, *Fit for Development,* Addison Wesley 2005