



Insightful Evolution™

Agile Application Integration

Damien Bastin
November, 2008

Agile Application Integration	3
The Background	3
Motivation	3
The Team	3
Project Management	4
Business Analysis	4
Environment Management	4
Technical	4
Briefly, The Details	4
The Star of The Show	4
Installation	4
Environments	5
The Master Database	5
Prioritisation, Task Allocation, and Tracking	5
Documentation	5
Data Migration	5
Interfaces	6
Kit	6
Vendor Engagement	7
Conclusion	7

Agile Application Integration

Successfully integrating 3rd party applications into your business is a serious challenge. There are several ways Agile processes can help to increase your chances of success, irrespective of the methodology employed during development.

Starting in March 2008, working for the Treasury Group within a large Australian bank, we applied practices normally reserved for Agile software development to a 3rd party application integration project. This paper outlines the reasons why, the issues we faced and how we tackled them.

The Background

Within any large company, the Treasury plays a critical corporate function. This is especially so within a bank. For this bank, the existing Treasury solution could not easily adapt to the continually changing nature of financial products. Over time it was becoming less and less relevant to the business. The Bank decided to replace the existing solution.

There are a limited list of vendors that offer IT solutions for trading the financial products typically used by corporate Treasuries. Through a tendering process lasting several months, this list was whittled down until one was chosen.

The chosen vendor had successfully integrated their product into several Australian sites using a project methodology based on the Waterfall model.

Correspondingly, the Bank had recently had some major success in reducing waste and increasing efficiency in their Software Development teams by employing Agile development principles. They were keen to see if some of those same principles could be applied to this project.

Motivation

There are compromises to be made by any enterprise when integrating a significant 3rd party application. This application integration project is no exception.

However, with a proficient team, and a willingness from both the vendor and the business to approach the project from an Agile perspective, we are able to successfully apply key Agile principles, such as¹:

- Individuals and interactions over processes and tools
- Working software² over comprehensive documentation
- Customer collaboration³ over contract negotiation
- Responding to change over following a plan

We try to keep doing what works, and have the courage to change what doesn't.

The ultimate goal is to reduce the total cost of ownership and maximise business value. The Bank sees using Agile software development practices as reducing waste, increasing efficiency, and more readily facilitating a culture of continual improvement. For these reasons, applying Agile software development principles to an application integration project was worth attempting.

The Team

As you would expect, there are differently skilled groups of people within the team, such as:

- Project Management (Project Finance, Resourcing, Vendor Management)
- Business Analysis (Domain Experts from the Bank and the Vendor)
- Environment Management
- Change Management and Training
- Technical (Testing, Development, Application Configuration Experts)

¹ Lifted from the Agile Manifesto - <http://agilemanifesto.org/>

² We emphasise this using timely automated tests.

³ In this case Treasury Front, Middle, and Back Office staff.

Project Management

As a start, the Bank identified an in-house Project Manager with a track record of impressive results with Agile software development. She was asked to identify whether Agile principles could be applied on this project, and how this might be done.

She identified that the process employed should be one that was continually adaptive. She also acknowledged that the team, including the Vendor, should be given the latitude to change and improve their processes over time.

Her prior success with Agile software development gives her a good idea about how to accept and control these changes and improvements.

Business Analysis

Experts from the business have significant time allocated to the project. Their "buy-in" is a significant factor in the acceptance of the application by the business. Assisted by Business Analysts with domain knowledge, these experts act as representatives for the rest of the members of their business unit. Specialist domain experts are rotated onto the project as functionality for their domain is implemented.

Environment Management

We have a great Environment Manager. Like a good supply sergeant, he knows how to play the game and get the goods. He helps us liaise with the various IT teams inside the Bank, to sort out hardware, networking, and backups for all our environments.

Technical

Members of this team have strong Agile software development experience, with a focus on Scrum and Extreme Programming (XP) techniques.

They are responsible for:

- Building and maintaining the development, testing, and production environments.
- Migrating existing data to the new system.
- Implementing interfaces to other applications in Java and C.
- Configuring the application.
- Continuously and automatically testing the entire system to ensure requirements have been met.

Members of the existing IT support team are part of the technical team. As a result, the non-functional requirements of robustness, maintainability, technical consistency, and quality are given special emphasis.

Briefly, The Details

The Star of The Show

The Application itself requires SunOS 5, 15G of RAM, and Sybase 12. Application configuration and data is spread across both the file system and the database and it takes around takes 20 minutes to perform a full backup and restore of the database.

The combination of all these things limits our ability to freely apply Agile best practice. However, dealing with technical obstacles like these is something every vendor integration project is faced with.

Installation

Initially, the application was installed over a period of 3 weeks. Technical consultants from the Vendor and an Agile software developer from the Bank worked together to install the application.

We had a goal that a team working on a stream⁴ would have their own environment. The Vendor was clear from the start that 3 weeks is the typical timeframe required to install the application. We realised that taking this long to create each environment would be far too costly.

As a team, we spent the following 3 weeks encapsulating as much of the manual process as possible into automated scripts. The scripts enabled us to reduce the installation time from 3 weeks to 30 minutes!

We use a continuous integration (CI) build containing a set of installation "smoke" tests to verify the the installation process whenever a change is made to the system,

⁴ A stream is just a group of related tasks.

allowing us to quickly and confidently deploy updates through to the testing and production environments.

Environments

Each work stream is allocated one of 14 environments. There are environments available for analysis, application configuration, interface implementation, data migration, continuous builds, testing, and training.

In up-front dollar terms, the number of environments is a large expense for the project. However, coupled with the fact that each environment can be easily refreshed, they are the key factors in enabling:

- Isolation of changes made by one work stream from other streams, encouraging innovation and change.
- Greater productivity through the ability to develop functionality in parallel.
- Sand-boxed Continuous Integration (CI) builds.
- Increased system quality by providing users and analysts access to the system when they need it.
- Ensuring the system integrates properly with non-development environments.

The Master Database

As mentioned above, the application makes use of a relational database to store application configuration and data. In order for the team to record, test, and share changes to the application, we regularly commit database backups. Using these backups, we can rebuild our "Master Database" at any time.

The Master Database changes are available to each environment. Each stream has a choice as to whether or not to absorb each change or not. Database updates to each environment occur as needed, usually once every few days.

Prioritisation, Task Allocation, and Tracking

We divide large problems into smaller, more manageable tasks that try to follow the I.N.V.E.S.T. guidelines⁵.

Prioritisation of team tasks is done in a typically Agile way. It is based on picking important requirement, then analysing how to implement the simplest example of a product feature that meets that requirement.

We prioritise tasks in order to get the skeleton of the functionality in place as soon as possible. We put more "meat on the bones" during subsequent improvements to the functionality, until we have a completed solution. A lot of feature implementation consists of simply configuring the Vendor's application to match existing business practice.

In some cases, we perceive that some features have a higher risk of failure. This is a consideration only when we have to decide the relative priority of two contenting features of equal business value.

An estimate of how many prioritised tasks can be completed in one fortnight is made by the team each fortnight, with a view to creating a meaningful release every 2-3 months⁶. Importantly, everyone in the group signs up to complete these tasks. Our progress through the tasks is reported to the business and management.

We have a really lightweight process to handle newly discovered tasks while underway (by recording and prioritising them accordingly)⁷.

Documentation

Light, adaptable, highly visible documentation is invaluable to our project.

We accepted that we would be learning lots about the Vendor application during the life-cycle of the project, and that the documentation supplied by the Vendor would not provide everything we would need to operate and support the application.

For information that has a lifespan of more than 1 month that will not be covered by application functionality, we record this information in a wiki accessible to the entire team, with no security restrictions.

Data Migration

Every application integration that replaces an existing system will have an element of data migration.

⁵ see writings by Mike Cohn for more on this

⁶ During software development this timeframe would ideally be reduced to every month or so. In this case, it is difficult to produce a meaningful release in this timeframe owing to nature of the integration.

⁷ We don't ignore new issues just because they throw out our original estimates. We adapt.

The goal for our team is to minimise the amount of data that we will need to migrate on the "go-live"⁸ night. Given the amount of data we are migrating, leaving all the migration to a process that runs at the last minute would be a big risk.

This topic requires a lengthy discussion in itself, but the approach to data migration we have taken can be briefly outlined as follows:

1. Try to minimise the data that needs to be migrated. Are you sure you really need all that data? Replacing an existing system is a good opportunity to clean house.
2. Categorise and deal with the migration data according to how volatile it is. We have three buckets.
 1. Data that is almost "static" and **never changes** (e.g. Countries that the Bank trades with). We have tried to get as much "static" data as possible into source control as part of your Master Database data (see [Master Database](#)).
 2. Data that **sometimes changes** (e.g. Public holidays for most stable countries are set 30 years in advance, however not every country is stable). In some cases, we treat data like this as if it were an Interface.
 3. Data that **always changes** (e.g. Customer deals). For this, we tried to find an ETL (Extract, Transform, Load) tool that allows us to run automated tests in a timely fashion.

All the code we write as part of this process is continuously, automatically tested.

Interfaces

Every application integration that replaces an existing system will have to also replace the existing application's interfaces to other applications within the enterprise.

On this project, we have 25 application interfaces to replace. Some are inbound, some are outbound, and some are synchronous request-reply.

The project has a goal to minimise the impact replacing the Treasury system has on the other enterprise systems. This means that we try to replicate the existing evolutionary and inconsistent solutions. The project is consciously avoiding a move to an Enterprise Service Bus (ESB) architecture at this time.

Inconsistencies in the existing solution are minimised by using an EAI (Enterprise Application Integration) framework that abstracts away transport and payload specifics as much as possible. Application interface implementations are then reduced to finding solutions to different transformation challenges.

All the code we write as part of this process is continuously, automatically tested.

Kit

We have the best hardware we can possibly get.

Often the people that are responsible for expenditure on equipment are not the same people responsible for expenditure on human resources. We have gone to great lengths to impress upon people the real cost of waiting for CPUs, networks, and disk drives.

Importantly, we have a high availability of pairing work stations. This setup consists of two keyboards and two mice attached to a computer with a large monitor⁹. This encourages communication and collaboration between team members. These stations are available for every team member to use, not just developers.

⁸ We have taken to calling this "Fight Night", or probably more accurately "Fright Night"

⁹ see Figure 1 - Pairing Workstation

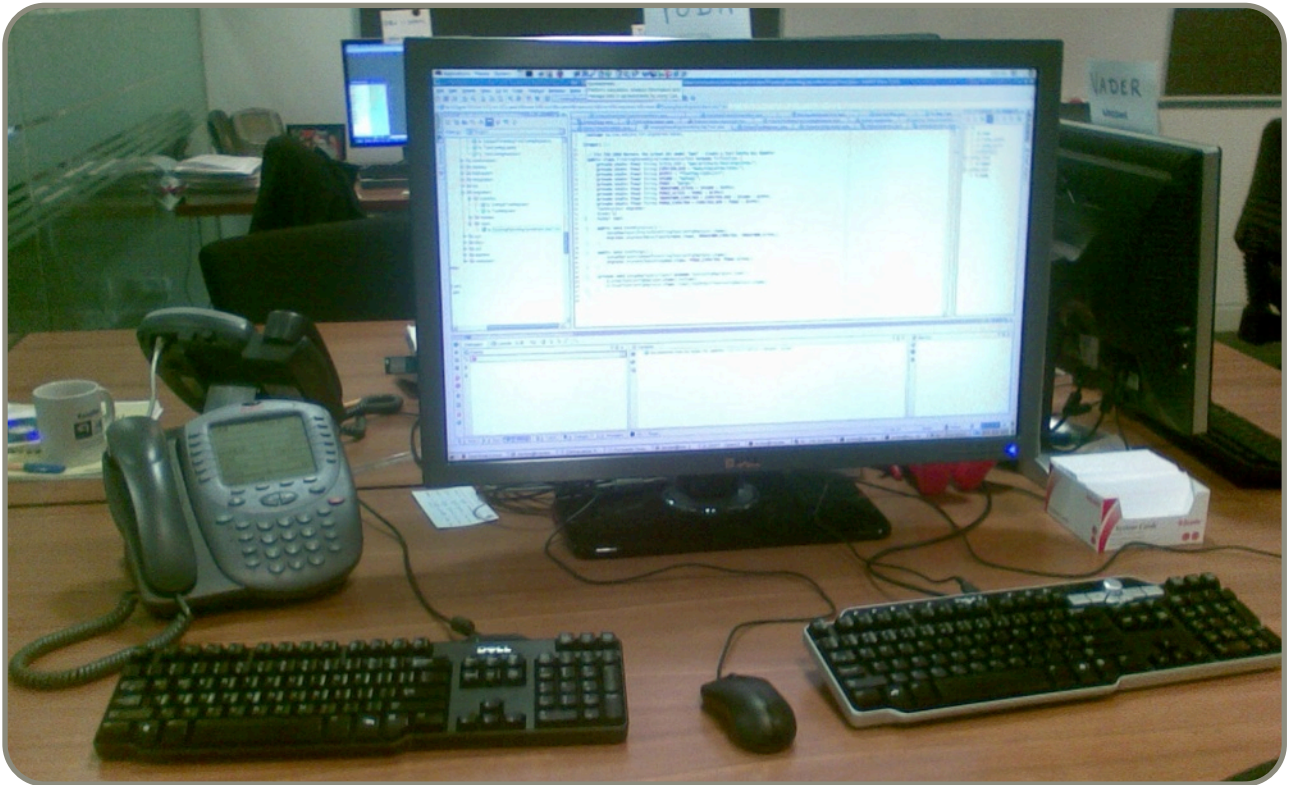


Figure 1 - A pairing workstation. These are used by all team members (not just developers) to make collaboration and communication easier. Two keyboards and mice are attached to a fast computer with a large monitor.

Vendor Engagement

Ever-changing scope, people, and timeframes are facts of life. Ideally, the agreement with the Vendor would be one that represents the realistic nature of integration projects like this one.

Unfortunately the contract with the Vendor is still very traditional. It gives either party little room to move, and does not easily accept the novel nature of application integration. It does not accept that things change as people learn over the course of the project.

There is a growing amount of literature about legal agreements that more accurately reflect the nature of large IT projects. Under the label Agile Contracts, more and more people are taking advantage of Agile processes in order to properly apportion risk and to maintain a healthy relationship between parties over the lifetime of an IT project.

Conclusion

When you don't have complete control over an application, integrating it into your business can be difficult.

By adapting Agile software development practices on our project, we have increased our chances of a successful outcome.

In practice, this means working within the boundaries prescribed by the Vendor's application to apply key Agile principles to the maximum extent realistically possible.

For information on how we can help you leverage software in your business, please call us on **1300-855-916** or email contact@leapstream.com.au

Full Contact Details

Australia

ph: 1300-855-916
fx: 1300-855-917

International

ph: +61 7 3129 2049
fx: +61 7 3112 1943

Snail Mail

GPO Box 1747
Brisbane
QLD 4001

Office Location

Level 6
Qantm House
138 Albert St
Brisbane
QLD 4000