

Experiences from Agile Projects Great & Small

Craig Smith*, Paul King**

* Suncorp, Brisbane, Australia.

** ASERT, Brisbane, Australia.

Emails: craig.smith@suncorp.com.au, paulk@asert.com.au

Abstract

Over the last five years we have been relentlessly applying agile practices on a number of projects, great and small, with decent success. These successes, however, have not been achieved without challenges and lessons learnt along the way. This report identifies some of the important practices we have learnt and specifically highlights examples from a number of different software development projects of varying sizes within this period and within the same organisation.

Keywords: agile, Extreme Programming, iteration management, Scrum, software engineering, XP.

1 Introduction

Agile projects almost always result in a successful outcome when the project team applies the principles and practices with enthusiasm, understanding and consistency. Any project in any organisation has to deal with unanticipated challenges. The challenge is to overcome these and carry the lessons to future work.

In this report we look at a number of aspects that make up a successful agile project and provide some real world examples of successes and challenges that resulted from them.

In particular, we will provide experiences on a number of projects that we have had the good fortune to work on over the last five years within a large financial services organisation [1]. These projects were great (including a corporate document generation system and treasury replacement system) through to medium and small (including an insurance policy processing module and banking batch payment processing system enhancement), amongst others.

2 Team Practices

Well executed team practices are an important part of a successful agile team. The daily standup, iteration kickoff meeting and retrospective are the main opportunities for the entire team to get together to communicate status, expectations and project health. Setting a meaningful length for iterations and releases and a team understanding for the definition of done

allow the team to understand and meet project milestones.

2.1 Daily Standup

The daily standup (also known as the daily scrum) is an opportunity for the team to get together at a specified time every day and briefly communicate project status.

In our experience an effective standup meeting should:

- Be held at an agreed time every day (with no exceptions), with the entire project team.
- Open to anyone external to the project interested in project status to attend and observe.
- Held standing up in a circular shape.
- Answer the questions “what did I do yesterday”, “what do I plan to do today” and “what impediments do I need assistance in overcoming”
- Be attended by all, with a precedent set that if any team member cannot make the daily standup time for any reason that they notify the iteration manager before standup commences.
- Not last longer than five to ten minutes (depending on the size of the team).
- Use a talking stick or other object to pass around and denote who is speaking and has the attention of the floor.
- Take topics that warrant an in-depth discussion off-line.

Distributed teams (where time zones allow), should benefit greatly from the standup as a defined opportunity to communicate. One scenario we have encountered is that teams can fall into the habit of the “daily sitdown”, especially when huddled around a telephone. This practice can be hard to fix if the habit is already entrenched, as it is difficult to enforce the standup behaviour to team members on the other end of the telephone

Larger teams, especially those that have a distinct technical team within a larger team, can benefit greatly

from a “post-standup technical meeting”. This gives the opportunity for more technical discussion of the current status of storycards being worked on, as well as allowing the team to sign-up for new work and rotate pair programming teams.

2.2 Iteration Kickoff

The iteration kickoff (also known as the iteration open) is a brief meeting (45 minutes maximum) to allow the project team to get together and align goals and expectations for the upcoming iteration.

The kickoff usually replaces the daily standup meeting on the first day of the iteration and is ideally chaired by the iteration manager with input from various members of the team. Rather than using a PowerPoint presentation, we encourage the key items to be added to the project wiki (allowing collaboration when preparing for the kickoff as well as team members easy access to the detail after the event). Ideal items for the agenda include:

- An overview of iteration and release goals.
- Key dates, calendar items and resource availability.
- Key metrics such as discussion of burn down charts.
- Status updates from key project areas.
- Short demonstrations of functionality delivered in the last iteration.
- Project updates (the types of material that may traditionally be discussed in a team or project update meeting).

For small or distributed teams, the iteration kickoff may be the only time the entire team gets together, therefore we have sometimes found it beneficial to hold the retrospective at the same time.

On one of our small projects, the combined session was held mid-week to accommodate the availability of the wider project management team. For each one-week iteration, a goal was specified, and the outcome of that goal was demonstrated at the end of the iteration. Regardless, it is imperative to ensure that the team is getting communication value from these meetings and that the team leaves with a consistent view of the upcoming iteration.

2.3 Retrospectives

Retrospectives are a get-together of the entire team at the end of an iteration to observe achievements and shortcomings impacting them as well as the opportunity to adapt appropriately. [2]

Retrospectives are best coordinated by the iteration manager or an impartial agile coach. We have observed that when coaching a new team, you need to build

confidence that input is valued and ideas will be acted upon. We have also noticed that when held informally (such as in the team area around a rolling whiteboard), the retrospective tends to have a more collegiate atmosphere and be viewed more positively by the team. We have found that a team discussion running through all of the items on the board further encourages participation.

We have used many different alternatives to correlating the information for a retrospective, but regardless of the breakdown you use to encourage feedback it is imperative that you allow team members to suggest actions for improvement as well as record feedback as it arises during discussion.

Retrospectives are difficult to coordinate with remote teams. Whilst feedback can be sought prior to the main meeting, we have found it more beneficial for remote teams to hold separate retrospectives in their respective locations, and coordinate the findings after the conclusion of these meetings for the entire team.

2.4 Iteration & Release Length

A consistent iteration length is imperative to building a team that can maintain a high velocity. The accepted standard for iterations appears to be agreed as two weeks, although we have witnessed one week iterations work extremely well on very small pieces of work, especially to deliver confidence and status to customers and management.

On one of our medium sized projects, the project had initially committed to four week iterations, firstly to alleviate the issue that the analysts need more time to collect requirements and secondly to use the last week for testing and deployment. In this case, we suggested moving to a two week iteration, which actually alleviated the load on the analysts. As for the final week, we observed that quality dropped as developers left testing for the final week (and thus when tests failed, the product was not ready for deployment anyway). Moving to a two-week iteration and enforcing testing as part of each storycard, alleviated this strain. Deployment was picked up as just another activity required in the next iteration (iteration plus one).

Similarly, release length should always be determined based on the number of iterations required to deliver a piece of functionality that is of enough value to the customer that could be deployed to production. We have seen many cases where the release length is guessed rather than based on storycard estimations, which too often leads to release functionality that is incomplete (and thus rendering the release structure redundant).

2.5 Definition Of Done

The definition of done allows the team to have a consistent understanding of what the completion of an activity or storycard means.

Typically, a developer will “sign on” to a storycard by talking to the customer and tester and agree and understand the boundaries and acceptance criteria of the card. At card completion, we have found the process of functional and technical signoffs useful to pick up any missing requirements as well as an opportunity to review code quality, refactoring, test coverage and documentation completeness (all of which should be met to achieve done). The definition of done ideally should also include the addition or acceptance of tests by a project testing resource, and agreement of any outstanding or newly identified functional requirements or code refactorings that should be spun out into new storycards.

We can usually identify teams that understand the meaning of done by observing that:

- The code quality is optimal.
- Storycards are often not left in a state of incompleteness across multiple iterations.
- Testing backlog is low.
- The customer is has a good understanding of the status of requirements.

3 Software Development Practices

Agile methodologies such as Extreme Programming (XP) [3] help developers productively deliver high-quality features to their customers on an on-going basis. Most agile practitioners would agree that using more of the agile practices on their projects is likely to raise the quality of the developed software but there would be diverse opinion about which of the practices are the most important and also which deliver a better return on investment.

3.1 Releases & Deployment

Agile teams value highly the practice of delivering useable software to production. Many of our teams have automated their build and deployment processes to the point where every time they change the code base (which can be many times per day) they can produce the artefacts suitable for performing a production release. In addition, they could run a suite of tests to gain confidence that the artefacts are release ready.

Striving for this goal has its problems. Firstly, many teams may not realise that the artefacts they produce aren't quite ready for production. Manual steps, such as testing, deployment or downstream integration, dramatically alter the feedback cycles within an agile ecosystem. The key issue for teams to tackle is to be sure they have a well-informed understanding of how

much of the complete deployment process they have automated. They can then work on reducing the administration debt associated with tracking the different versions of their software that span between development, testing and production environments.

Depending on the nature of a project, it may not always be possible to automate as much of the deployment process as an agile team might like. Some of the tricky issues which might be faced include:

- Integrating with other parts of the organisation which may be less agile.
- Working with very complex products which span across multiple environments.
- Dealing with distributed teams which may suffer from communication barriers.
- Development environments which differ significantly from production environments.
- Dealing with hard to test vendor supplied products or solutions.
- Dealing with cross-functional boundaries during deployment where different teams may administer operating systems, application servers, databases, security and so forth.

As well as ensuring the team has an excellent awareness of its deployment process and striving for high levels of automation, we also use tools like Tableaux [4] to manage the deployment process across environments in a disciplined way. Such tools allow the deployment process to be represented in an understandable and visual way, help streamline automation of the processes and help manage the differences that occur between environments.

3.2 Developer Practices

Some agile methodologies have little to say about the actual practices developers should use to strive for high quality solutions. We have found that by adopting many of the tenets of Extreme Programming (XP) and with an attitude of continuous improvement, teams can normally produce high-quality solutions with minimal reduction in productivity.

Some of the techniques we use include:

- 100% code coverage from unit tests to eliminate untested code and avoid team tensions about what should or should not be tested.
- All production code paired and test-driven as a practice to keep the code simple and maintain consistent quality.
- Minimal design up front but an appreciation for when such design is appropriate.

- Full continuous integration with multiple builds triggered as code changes in order to improve feedback.
- Daily pair rotation to share knowledge throughout the team.
- Continuous improvement including retrospectives, pair programming and shared code ownership.
- High levels of automation.
- Make certain kinds of tests disposable because they could be easily generated from production code.
- Automate certain kinds of otherwise tedious to test functionality, including immutability, cloneability and null checking.
- Autogenerate certain kinds of random test data to simplify tests and triangulate over time.

We regarded these practices as what was necessary to “turn the dials to ten” but found that team morale and productivity remain highest when we continually tried to improve even further.

3.3 Developer Innovation

To enable a team to “turn the dials to eleven” [5], it is imperative that the development team turn to innovation and quality metrics to assist in delivering quality code and a consistent pace.

An on-going problem we have encountered is the amount of source code duplication. When a bug is found, it often means changing code in several places with great confusion when not all places were fixed. To combat this, we have used the Simian duplication detection tool [6] with a threshold of four lines. If any four consecutive lines in any production or test source file are duplicated, the build would break and we would refactor out the duplication.

Using automated checking tools such as Checkstyle [7], cyclomatic complexity can be set low; effectively prohibiting error-prone code constructs such as instance nested looping statements, conditions or try catch blocks. Furthermore, methods can be limited to approximately 10 lines and classes to about 100 lines (with potentially slightly different metrics for test and production code). These types of metrics force a refactor of any significantly complex classes. Furthermore, we have developed our own metrics plugin [8] which enabled us to have exactly the same metrics rules in place and ‘live’ within the Integrated Development Environment (IDE) as we did within our continuous integration build.

We also increased our productivity by evolving a small but carefully constructed framework of utilities that allowed us to:

- Efficiently use inversion of control.
- Efficiently integrate hard-to-test third party libraries into our system in a way that facilitated testing.
- Increase our productivity when writing mock style interaction tests. [9]

4 Project Characteristics

Agile principles are of benefit to any project but the individual project characteristics can affect execution and the strategies used. Duration and type of project can have a major influence, as well as the approach and constraints relating to design.

4.1 Project duration

The size or duration of a project should have little bearing on whether agile practices are considered. In most cases, it is the disciplines of the project team and their ability to adapt to the project space that dictates the successful adoption of agile practices.

Our experiences show that small projects can be very successful, by tweaking the practices to suit the circumstances. A small team (1-3 people) for a small duration (2-6 weeks) should consider one week iterations, as they help enforce short iteration cycles and regular customer feedback. There is sometimes a perception that there is a need for a large overhead, such as a separate or dedicated scrum master or delivery lead, but our experience it is that very easily consumed by the team on a project of this size, with some planning and commitment.

Similarly, medium and large size projects can also be successful. Optimising communication is often the most difficult part of a large project. Attempting to keep the team in touch with overall progress through formal channels must be balanced with the high level of informal and in-depth conversation that make agile teams successful.

Large teams that have a high level of agile expertise can sometimes suffer through differing points of view on the level of practices that should be undertaken by the team. Similarly, large teams that have a small or mixed amount of agile expertise can suffer through adoption of practices and unwillingness to adapt processes.

4.2 Project type

The majority of successful agile projects we have encountered have been traditional application development projects. Software engineers, especially those that appreciate the skill of quality and well-crafted software [10], are those that tend to support and encourage the types of principles and practices that the

agile methodologies deliver. It is these projects that tend to have a greater success level with agile adoption.

Other types of projects (such as those that have a heavy integration or infrastructure effort) make full agile adoption harder [11]. Sometimes the application development stream of the project will push agile practices that seem foreign to other parts of the team. Resistance can be from application vendors or other third parties assisting with the project or from infrastructure deployments that for various reasons need to follow a strict plan to successfully deploy or upgrade hardware.

There are many ways we have used to counter these issues:

- Agile practice education can highlight the benefits to all members of the team.
- Vendor contracts can be written in such a way that demonstrates the risk is shared between all parties (which is normally the case anyway).
- Third parties that must continue with traditional techniques can be worked around in the agile plan.

4.3 Design constraints

One of the tenets of agile software development is building the simplest things that can possibly work. In agile circles, this commonly described as eliminating “Big Design Up Front” (BDUF) [12]. However, BDUF does not necessarily equal “no” design, it should just mean the appropriate level of design. This is often referred to as deferring design decisions until the last responsible moment.

In many cases, writing software that meets the simplest design required right now can have immense benefits as the project evolves and the requirements become clearer, with a simple design being much easier to refactor as appropriate.

However, on some projects, we have found that some up-front design is appropriate, if not, essential. In some cases, a simple whiteboard discussion by the development team at the start of a release or a piece of work to get a common understanding of direction is all that is needed. In other cases, a full design specification for a piece of the application, for example a client-facing web service API, has been essential in ensuring the design met all of the key requirements. In our experience on projects such as these, an upfront design of the XML meant customers could start developing earlier and the amount of required integration testing was greatly reduced.

5 Equipment, Tools & Facilities

Equipment, tools and facilities are often overlooked, but essential, characteristics to the delivery of any project.

When applied to agile, there are some clear benefits to the velocity of the team in getting the balance of these correct.

5.1 Distributed Teams

Distributed teams are increasingly becoming a fact of project life, with team members being geographically dispersed (between countries, cities, buildings or floors). Furthermore, teams are being technically dispersed, with team members working in the same sub-team or with the same technical skillset being deployed in different locations.

For the most part, nothing compensates more for this than travel. Experience has shown time and time again that getting team members face-to-face increases honesty and morale and overall team velocity. To supplement this however, technology can be used when the team has to work remote, through the user of tools such as collaboration wikis and iteration management tools, instant messaging, video conferencing and remote desktops. Video conferencing and shared collaboration tools are essential for involving remote team members in activities such as iteration planning and retrospectives.

Remote pair programming is one technique that we have attempted. Using a shared virtual desktop and telephone headsets, team members were able to collaborate on code and tests. The maturity of the team has a major effect on the success of this technique. Team members that are comfortable with the concept of pair programming usually have little problem adapting. For those new to the practice, co-location and/or coaching are required. Alternatively, we have seen the same technique used to great effect for remote buddying, when a senior technician is paired with a less-experienced team member to assist with knowledge transfer, monitoring and review.

5.2 Collaboration Tools

The base agile tool often prescribed for agile planning and progress tracking is index cards pinned to a wall to show current status. Commonly, even with a co-located team, there is a need to have the story wall and other visual indicators virtualised so that external stakeholders, distributed team members or even co-located team members due to seating arrangements can see and interact with the story wall.

There is a myriad of different tools with varying levels of functionality and price points available in the agile management space. Tools such as Microsoft Excel and Microsoft SharePoint can be used on small to medium projects, although they require discipline and setup to work effectively. We had great success in the past with the free and open source XPlanner [13] on projects this size, as it simply allows storycard tracking and monitoring with a design that is useable for all members of the project. Unfortunately, this project seems to have gone stale, with little activity since 2006.

More recently, for medium to large projects, we have been using the commercial tool Atlassian Jira [14] in conjunction with the GreenHopper plugin [15] to great effect. This combination presents a reasonable compromise even though Jira's heritage as a traditional bug tracking system is always evident. GreenHopper emulates planning, story wall and chart functionality and does a good job extending Jira functionality without affecting the basic Jira functionality. One benefit we have brought teams using this tool is that they can truly track their entire project in one place, including stories, project activities, defects and bugs as well as project risks and issues.

Collaboration wikis, such as Atlassian Confluence [16], have in our experience proven to be extremely beneficial in ensuring that project knowledge and status is recorded in an easily accessible and searchable location. Newcomers to these tools often need to be coached and encouraged to "add it to the wiki", but as knowledge starts to accumulate we have found that the value is usually realised. Especially useful is using the wiki for tracking project meetings such as retrospectives as well as support notes. We have also found it imperative for the team to periodically "refactor" the wiki to ensure the detail is current and easily findable.

5.3 Facilities

The agile principles strongly encourage the concept of a co-located team. Whilst in many of our projects we have been able to achieve close to a fully co-located team, the reality of a large organisation in conjunction with fixed facilities have often meant the project team has had to undergo numerous relocations and seating arrangements that were less than ideal.

Often, our teams were faced with squeezing into meeting rooms, legacy cubicle farms and odd-shaped curving desks that made pair programming difficult. As a result, a level of ingenuity was often required to "knock" the team area into space, as well as the use of basic tools such as screwdrivers and clamps!

On the flipside, some of our projects have been given more suitable accommodation, but more isolated from stakeholders and peers in the organisation. However, an environment that allows the team to communicate effectively and remain focused at the expense of communication with the wider organisation is often preferable.

6 People & Roles

People are the most important asset on any project team for obvious reasons. However there are sometimes challenges to ensuring that an agile team moves from a group of team members to a team that is high performing.

6.1 Influence

A major facet of the agile methodology is to embrace change, something that many individuals have difficulty in adapting to, especially if they believe there is nothing significantly wrong with traditional approaches to delivering software.

Training and coaching are the obvious and most productive ways of training and teaching staff new to agile, with training programs such as the Agile Academy [17] becoming more accessible. However, there is no substitute to real project experience, so a team with experienced agile practitioners and/or an agile coach that can demonstrate the value of the practices is by far the preferred approach.

It is interesting to note that in many cases reluctant adopters will continue to be reluctant regardless of the level of training or coaching that is in place. This can be relatively destructive and demoralising to those that are attempting to drive the agile adoption. On the flipside, however, those individuals often are adopting and even supporting many of the processes without realising the level of change they have made, especially in the circumstances where they are surrounded by a team of experienced agile practitioners.

On one of our medium sized projects, we were assisting a team that was new to agile techniques, as well as being distributed, therefore new techniques such as test-driven development and pair-programming were requested by management after successes on previous projects but were hard to introduce as the team did not have appropriate mentors or coaching in each of its locations. Furthermore, we observed that if the concepts are introduced but not reinforced with this technique initially, it is very difficult to enthuse the team to give them a second chance.

6.2 Co-Located Customer

A co-located customer is an extremely important part of any successful project as it allows the project team to collaborate effectively and change direction quickly when requirements are not understood or the delivered software is not as expected. The reality on many projects however, regardless of the importance or complexity of the project being introduced, it is difficult or even impossible to get the customer co-located.

In those scenarios, one option is to introduce the notion of a proxy customer, who is empowered by one or many customers to make decisions on their behalf. This role can work well if they indeed can make decisions and have enough business knowledge to make a decision with confidence.

In other cases, a team may be forced to work with a remote or work-overloaded customer. For medium to large sized projects, this can be extremely difficult and the use of a proxy customer is highly encouraged. For small projects with short iterations, a remote or

overloaded customer can work if they are available for a weekly face-to-face retrospective and acceptance session, coupled with the ability to instant message or telephone to clarify requirements.

6.3 Embracing the Wider Team

Many of the XP practices focus directly on the development and subsequent delivery of software; whereas the Scrum methodology has a large focus on the management and lifecycle practices of an agile project. Implied, but not necessarily directly referenced in these methodologies, is the role of the wider team resources such as testers, business analysts, architects, and infrastructure and deployment specialists.

In recent times, a lot of focus in the agile community has been placed on the role of testing [18]. Traditionally the testing area has been viewed by some as the place where the code gets thrown after the developers have finished with it. In agile, the role of development and testing has become much closer. In our experience, the closer you can get the developers and testers to development and resulting completion of a storycard, the more successful you will become. Many of the agile texts relate to the developer side of testing, but the skill of quality assurance and user acceptance testing should not be overlooked on any agile project. Also important, is the team agreement on where the role of testing fits into the “definition of done” in relation to the completion of a storycard.

Business analysts are often confused or blended with the role of the customer. In this case, it is extremely important to identify the role of the proxy customer or the product owner, so that business analysts can focus on the analysis and articulation of requirements. Architects also sometimes have a confused role, especially on teams where the design is evolving. The ability for an architect to be involved and work with the development staff as opposed to dictating to them from afar is extremely important.

Infrastructure staff (including system administrators and database analysts) and deployment staff are traditionally not co-located with the project team, especially in large organisations, however due to the nature of software development these resources are usually required on an ad-hoc basis with very little warning. The optimal approach is to have these resources with the project team, but when this is not practical, it is desirable to make available a dedicated resource who works on an as-needed basis.

Change management is a role that is often overlooked until the end of a project, however on large projects the closer you can place this effort to the development of the system, the more successful the training and resulting process changes will be. Change management following each release should be the desired minimum, with iteration or even storycard change management desirable.

Usability testing can often be an area that suffers on agile projects, especially as the short iteration cycle and simple design practices are blamed for the inability to adequately apply usability principles. Some of our small to medium projects have recently had a large usability push as the benefits have been realised of incorporating usability into the process of parallel track development and the use of other techniques such as shared values and personas amongst others.

6.4 Attracting Quality Staff

A benefit of a project team that has a clear commitment to agile practices and a development team that is following good agile practices to the extreme is the high level of expertise of candidates that apply for positions throughout the project. Staff rotation through the project also allows the agile message and many of the practices to start propagating throughout the organisation, if done effectively.

In our experience, most candidates come via word of mouth from conversations with members of the team. The result for many of our projects was a team that was “the place to be”, both within the organisation as well as the wider community. In some sense this then became self-fulfilling as many innovative ideas became possible due to the high calibre team. This also had the flow on of a relatively low turnover of staff.

Third party vendors can sometimes be problematic to deal with; if they aren’t very agile, it can be hard to align their deliverables and feedback cycles with our own. However, we have had positive experiences when we took the time to explain our approaches to them when they were on site. A number of vendors and consultants we have worked with have been excited about many of our practices and demonstrated successes and as a result have taken on some of these practices internally themselves. This has been exciting to hear about but also has resulted in more aligned interactions with the vendor.

7 Change from the Bottom Up and the Top Down

Early agile deployments started from the bottom up, with practitioners leading the charge. In recent times, many organisations have pushed agile from the top down.

7.1 Bottom Up

One of the reasons for the adoption of XP by many teams is that, being developer focused, a new methodology was pushed by developers eager for a change and better ways to deliver software. The fortunate part of this approach is that many of the practices that have become popular as a result of agile adoption, such as test driven development, refactoring, continuous integration and simple design, are now

accepted as best practices for software development, regardless of the methodology.

Introducing agile from the bottom up can be extremely successful if you have the buy-in of the majority of the team at the lower level, which is certainly our experience with successful agile projects over the last few years. The willingness of the project manager is also important to shield the team from the traditional outside forces.

One of the biggest issues is getting customers and stakeholders to understand the new approach. For customers, knowing that they will see software in the development stages (and not in a completed stage) can be an eye opener. Management and stakeholders also need to be educated that a well run agile team will be providing updates in a more open and honest manner. They need to learn to understand the new level of open communication and have faith in the team that they have a good understanding of the current issues and similarly will ask for help up the chain when required.

7.2 Top Down

Agile in the enterprise is a growing trend for organisations that are looking to react more quickly to market forces [19] as well as those looking to eliminate waste in the development process.

Agile adoption driven from a senior management level has the benefit that the organisation has a clear commitment to change and deployment across all areas and skillsets. This is certainly an area which we are observing more frequently on our current projects. Furthermore, our experience is that top down adoption is more successful when deployed in conjunction with bottom up.

For individuals new to agile, this approach means that the agile cultural change is forced upon them, in many cases via training without practical experience. It is imperative that this change is managed closely, especially to those who may resist the change.

For existing agile practitioners, especially those who may have been driving agile from the bottom up, corporate initiatives may impact the way their teams are currently approaching agile, especially as team members newly trained in the corporate agile approach may start questioning techniques that have not been fully articulated in training programs.

8 Conclusions

Adopting agile practices involves many interconnected facets with many exciting and challenging issues to face. While many common themes arise out of adopting agile practices, we do not believe that there is any single predefined template that can be rolled out verbatim to all agile projects.

Our experiences over the last few years on many different projects vastly different in nature and size, both great and small, has led us to fully appreciate the benefits of adopting the various agile practices but also reminded us that the way to apply these practices is different for every project. Furthermore, we believe the best approach for successful outcomes on an ongoing basis is to learn from your own past experiences as well as the experiences of others.

Acknowledgments

The authors wish to thank Suncorp management, members of the EasyDoc, Guidewire Policy Center Pilot, Treasury Systems Replacement and Shared Java Apps teams as well as other Suncorp staff who assisted the authors or were involved in the described projects or related activities within Suncorp.

Suncorp is one of Australia and New Zealand's largest diversified financial services providers, supplying banking, insurance and wealth management products to around 7 million customers through well-established and recognised brands such as AAMI, Australian Pensioners Insurance Agency, Shannons, Vero, Asteron and Tyndall, as well as Suncorp and GIO. Today, Suncorp is Australia's sixth largest bank and second largest domestic general insurance group, with over 16,000 staff. Suncorp has representation in 450 offices, branches and agencies throughout Australia and New Zealand.

ASERT is one of Australia's leading suppliers of development services, mentoring and training in Agile, Web Services, Web Applications, Java, Groovy and Grails.

References

- [1] C. Smith and P. King, "Agile Project Experiences – The Story of Three Little Pigs", Agile 2008 Conference Proceedings
- [2] E. Derby and D. Larsen, "Agile Retrospectives: Making Good Teams Great", Pragmatic Bookshelf, 2006
- [3] K. Beck and C. Andres, "Extreme Programming Explained", 2nd Ed, Addison-Wesley, 2005
- [4] The Tableaux Configuration Management Tool, <http://www.incanica.com/>
- [5] P. King and C. Smith, "Technical Lessons Learned Turning the Agile Dials to Eleven!", Agile 2008 Conference Proceedings
- [6] The Simian Code Duplication Detection tool, <http://www.redhillconsulting.com.au/products/simian>
- [7] The Checkstyle Project, <http://checkstyle.sourceforge.net/>
- [8] The Agile Plugins Project, <http://code.google.com/p/agileplugins/>
- [9] D. Koenig, A. Glover, P. King and G. Laforge, "Groovy in Action", Manning, 2007

- [10] R.C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", Prentice Hall, 2008
- [11] D. Bastin, "Agile Application Integration", http://www.leapstream.com.au/downloads/papers/Agile_Application_Integration.pdf
- [12] Wikipedia: Big Design Up Front, http://en.wikipedia.org/wiki/Big_Design_Up_Front
- [13] The XPlanner Project, <http://www.xplanner.org/>
- [14] The Atlassian Jira Bug and Issue Tracking Software, <http://www.atlassian.com/software/jira/>
- [15] The GreenHopper Jira Plugin, <http://www.greenpeppersoftware.com/confluence/display/GH/Plugin>
- [16] The Atlassian Confluence Enterprise Wiki Software, <http://www.atlassian.com/software/confluence/default2.jsp>
- [17] The Agile Academy, <https://www.agileacademy.com.au>
- [18] L. Crispin and J. Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams", Addison-Wesley, 2009
- [19] G. Smith and A. Sidky, "Becoming Agile: ...in an imperfect world", Manning, 2009