Enterprise Agile

Applying Agile principles to enterprise development

The interest in agile development practices in recent years has been a reaction to the need to make software development projects more focused on early and regular delivery of business value. In many ways this has been a healthy shift, with a spotlight being placed on certain corners of the development process and the challenge put: “how does this task or deliverable help with the creation of business value?”

However, as is often the case with new approaches, the pendulum can swing too far, and the inherent value in some current best practices can be lost along the way. This paper briefly examines the principles and benefits of agile approaches, then highlights the problems such approaches face when placed in an enterprise context and how these problems can be resolved, while adhering to agile principles, by introducing enterprise best practice to create an Enterprise Agile approach.

June 2006

by John Cheesman, Strata Software Ltd

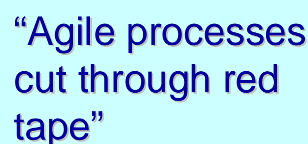
Project Management processes

IT projects frequently fail to deliver, either on time, on budget or at all. Occasionally this may be due to the software development itself or the technology being problematic. But it is much more likely to be due to problems with project scoping and the match between the delivered software and the actual business requirement.

Traditional waterfall approaches have attempted to solve this problem by “pinning down the requirements” and carefully scoping projects early on. This suffers from a number of major problems – (i) requirements and priorities change with time, so by the time the project delivers it is unlikely to meet the requirement, (ii) “paralysis by analysis” – the requirements are so extensive that they change during (and often because of) analysis itself, which means that analysis goes on and on (iii) technical risks to the project (e.g. technology problems, performance issues, security) may only become apparent very late in the project and consequently be very expensive to change.

Iterative approaches try to address these problems by breaking a project into a number of iterations, with each iteration being a mini-waterfall project. Rather than getting everything pinned down up front, the first iteration can start with a subset of the requirements, while the details of the requirements for the next iteration are fleshed out in parallel. Further, areas anticipated as being technical risks can be addressed in early iterations reducing risk for the project. Certain iterations are designated as releases, delivering software back to the business for review and acceptance.

Problem solved. Or is it?



Why Agile?

While iterative approaches are generally acknowledged as being an improvement, their overall efficiency depends on the type of “mini-waterfall” process being applied per iteration. Some iterative approaches identify a mass of development artefacts, documents, models, designs, tests and so on, that must be produced. These non-software artefacts can easily gain a life of their own, using up resources and causing confusion when inconsistencies arise.

Agile processes attempt to cut through the red tape and get everyone in the project, regardless of role, focused on the common objective of delivering software that provides business value as quickly and efficiently as possible.

Agile takes an iterative approach but forces the pace, shortening iterations to two or even one week, and paring down the deliverables and artefacts produced during the iterations. Each artefact is examined and its value challenged:

Do we need this document? Do we need this model? Do we need this tool? Can we write the software without it? Many activities and artefacts produced on iterative projects fail these tests and get pruned. The effect is to create a development process that looks something like Figure 1.

From an artefacts point of view it is highly software-centric. Business Analysts (BAs) communicate informally with the Business (Biz) to understand the business rules. They then communicate informally with

Copyright © 2006, Strata Software Limited

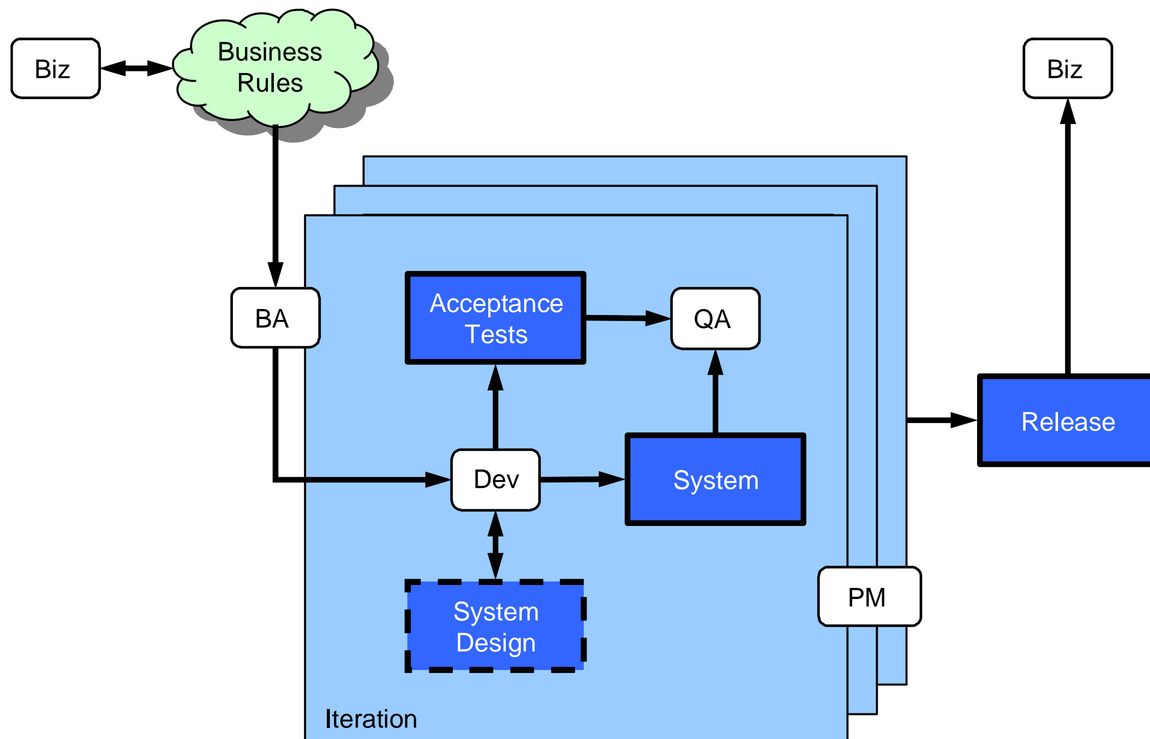


Figure 1 – Agile Process

the development team (Dev) to define the functionality of the system in terms of "stories" – small packets of functionality that can be developed in a single iteration. The software is periodically released back to the business for validation and this forms the main feedback loop.

Within the iteration Dev write acceptance tests for the quality assurance (QA) team to validate the software against. Dev will also write their own internal unit tests to ensure consistency of the code base across re-factorings.

Dev may also document certain elements of the system design, if this is deemed necessary, but due to constant communication and potential re-factoring the design necessarily remains a dynamic construct.

The project manager (PM) is responsible for the iteration and release planning, as normal, and progress is measured in terms of stories completed, and rate of completion.

This goal-driven and delivery-focused approach is fast and lightweight, and aims to establish a highly-responsive, customer-focused project, ensuring steady delivery of small increments providing business value. Does it work?

Agile was designed for small projects with co-located teams and UI-oriented systems (where functionality can be assessed readily through the user interface) and in this context it works very well. However, a number of problems can arise when applying it to projects that don't have the required characteristics, and particularly when scaling up to medium or large projects. We can categorise the problem areas broadly as:

- Requirements definition
- Scalability
- Enterprise context

Let's look at each of these in turn and summarise where enterprise best practice can usefully be applied.

Complementing Agile Approaches

Requirements definition

Agile approaches rely on informal communication of requirements, often using story cards or just a whiteboard. The idea is that by creating a rapid feedback loop and showing working software to the business at the end of each iteration, the need for any more formal requirements definition can be avoided.

Precision – informal communication is the life-blood of any project, but important business rule details must be documented to avoid miscommunication. Capturing these in a simple model ensures all the stakeholders are communicating, and that the rules are precise – *prior* to the software being developed. It is highly inefficient to wait for the end of an iteration to discover that there has been a basic misunderstanding of the rules. A symptom of lack of precision is the need for repeated analysis workshops.

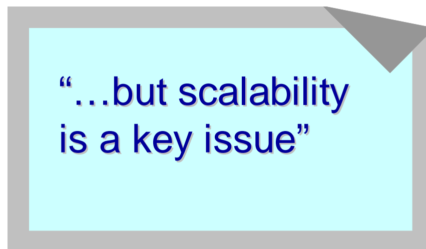
Stakeholder availability – relying on regular business acceptance as a mechanism for validating requirements is risky. Business stakeholders are busy people and their availability is often limited. Further,

Copyright © 2006, Strata Software Limited

iterations may slip meaning that functionality to be reviewed is not available on the planned day. These factors mean that business acceptance may occur much less frequently than the development iterations deliver increments. Augmenting software demonstrations with formalised models of business rules is a good way to mitigate this problem. Business rule models can be emailed around and reviewed in parallel by multiple stakeholders, catching problems before software gets built.

Single source of truth – identifying a single individual who can finalise the definition of a business rule can be difficult. Business rules may emerge during the analysis process and need reviewing by other stakeholders before they are finalised. As before, this definition and review process can be usefully supported by a small number of formalised and precise models.

Joined-up stories – agile development iterations are based on the selection of a certain set of stories. These are then implemented during the iteration and the functionality delivered. The goal of the delivery is to provide incremental business value, but this value can often be missing because stories don't join up properly. It is important that the collective set of stories selected for a given iteration supports a number of specific business scenarios. In order to do this, the stories must be based on a clear business process context.



Project scope definition and tracking – while joined-up stories based on business process scenarios are needed for iteration scope planning, similarly the project as a whole needs a clear functional scope definition mapped back to the business process. Agile developments tend to work with simple story lists and as development proceeds, stories get implemented, and progress is tracked against stories. This measures *development progress*.

But, as before, stories alone do not capture business benefit or provide a link to the business process supported. Project scope and progress tracking needs to be based on *business value delivered*. Projects can often be perceived to be progressing well by development measures (good velocity, large number of stories completed), but actually be failing to deliver on key business scenarios.

Acceptance – showing working software to the business is a good discipline, but using it as the mechanism for ensuring requirements and business rules have been successfully understood and implemented can be problematic. Apart from the availability problems mentioned earlier there are other issues:

- Reliance on the user interface – many systems integration projects have little or no user interface. For example, a data cleansing system may process data and remove duplicates. The business user can hardly check the database contents. What they can do is *define the required rules* and check representative examples.
- Rules can be complex – the verification of some business rules can difficult and best done against documented rule definitions rather than the working system. For example, distribution companies may use complex routing rules when scheduling deliveries. It is difficult for those rules to be validated by looking at the resulting schedules produced by a routing and scheduling system. Again, a small number of examples can be checked, but *examples do not define the rule*.

Scaling agile projects

Agile approaches were originally intended for small co-located teams working closely with the business stakeholder. Once we try to scale the approach things can start to break down.

Communication – probably the most important difference between small projects and larger ones is the ease and form of communication. Ten people in a single room can rely on informal communication and a couple of shared whiteboards. The daily stand-up keeps everyone is synch. All the developers understand all of the code. Life is good.

Once projects grow to even a modest size this idyllic picture unfortunately breaks down. Larger projects necessarily get broken into separate teams with different responsibilities. Having teams on different floors, even different rooms, can significantly affect shared understanding. Further, people take holidays, are off sick, leave the company, take another contract, and so on.

Relying on knowledge in people's heads, and communication via pair programming, while important, is not a viable enterprise approach. The solution is to get the key things written down and shared independently. The trick to remaining agile is being clear about what these key things are, and keeping their representation simple. We'll discuss this later as part of the Enterprise Agile approach.

Distributed development and off-shoring – the ability to partition system responsibilities and allocate ownership to different groups in different locations, including off-shore development, is an important strategic requirement for large organisations. This can be achieved by solving the communication challenges discussed above, but also by having well-defined and relatively stable architectures and development standards. These allow software dependencies to be clearly defined and understood and hence properly managed.

strata software
Copyright © 2006, Strata Software Limited

Collective ownership – a related topic to defining clear responsibilities is the agile principle of collective ownership. The idea is that anyone on the project can work on any piece of code. The principle is to ensure that everyone understands how the whole system works, to create less dependence on individuals, and to develop an ego-less attitude towards the code.

On large projects this is just not practical. With increased complexity and a greater number of technologies comes the need for increased specialisation. User interface developers have different skills from database designers. Security specialists are different to package experts.

Further, large projects often cross organisational boundaries and responsibilities. Changes therefore need to be managed in terms of well-defined software interfaces and organisational boundaries, with managed releases and deployments respecting those boundaries.

Application Architecture – agile approaches promote the concept of an emergent and evolving application architecture that is minimally sufficient for the functionality required to date. The idea is explicitly *not* to look ahead (at least not too far). Should subsequent iterations require an architectural change then it can be handled through re-factoring.

This approach generally does not scale. The larger the code base, and the greater the number of development teams, the more the need for standards. Introducing a major architectural re-factoring mid-flight on a large project is to be avoided at all costs.

For this reason, it is best practice in most iterative processes to address the major architectural qualities of a system early on and incrementally add functionality in subsequent iterations. Re-factoring can then be *localised* with less impact on other parts of the system. Focusing on functionality at the expense of architecture too early invariably costs more time and effort over the lifetime of the project. This does not mean that the architecture does not evolve, but that the evolution is managed on a sound base.

Enterprise Context

Notwithstanding the project-specific issues, there is also the wider enterprise context to consider. Few projects are islands. Most developments will need to dovetail with systems that implement overlapping business processes.

Enterprise Architecture – enterprise architecture concerns the definition, roll-out and maintenance of architectural principles and standards across the business process and system portfolio. All projects need to conform to these wider standards to support the wider IT strategy, even where, for that specific project, such standards may not provide a direct benefit. By taking too project-focused a view, agile projects can tend to exacerbate portfolio-level problems such as functional duplication, lack of reuse, differing artefact standards, information integration, process misalignment and so on.

> "Enterprise Agile incorporates the portfolio context"

The solution to this is to ensure that portfolio-level goals and project-level goals are properly aligned and that organisational responsibilities are clear. This means each project has a pre-defined set of *portfolio-level requirements* it has to meet.

Green field assumption – while many small projects may involve a high degree of new code development, most significant enterprise developments concern the integration, rationalisation or migration of existing systems and packages. Here, the vast majority of the code already exists, may be written in many different languages, or may not be available at all. In this context, architectural constraints and functional boundaries need to be based on a portfolio-level view of the problem, not just a project-level view (see also portfolio re-factoring next).

Portfolio re-factoring – the agile technique of re-factoring is aimed at the individual code of an application, but at the enterprise level the idea needs to be applied to system *portfolios* not individual systems alone. The current best practice – service-oriented architecture (SOA) – is just such a portfolio-level re-factoring approach, looking at the system portfolio as a whole and re-factoring systems into services over time.

Total cost of ownership – when looking at the costs involved with managing a system across its lifetime the maintenance and upgrade costs typically far outweigh the cost of initial development. It is therefore important that sufficient attention is paid to the maintainability of systems, not just how quickly you can get to first base. There are two aspects to this (i) system agility and (ii) system documentation.

Agile projects generally do not produce agile systems. System agility refers to the ability to make changes to the system readily, replace pieces, re-write pieces and so on. This system property derives from good internal structuring and dependency management using layering standards together with component- and service-based principles. Maintainability concerns are generally not uppermost in the agile designers mind

Copyright © 2006, Strata Software Limited

because there is usually no explicit "maintainability" story defined. This means that systems developed by agile projects tend to be more difficult to change later. By factoring in early on a number of standardised quality requirements for maintainability, the architecture and flexibility of an agile project can be greatly improved. This not only reduces the longer-term cost of ownership, it also assists with immediate changes of functional direction and content by providing a flexible systems framework to support such change.

Documentation is a classic issue. Everyone expects documentation to exist, but no-one wants to write it. Agile principles stress "working software over comprehensive documentation" and it is hard to argue against that, but some agilists treat this as an excuse to produce *no* documentation at all. As ever, a balance needs to be struck. In general, high-level summary documentation tends to be more useful than detailed documentation because a) it gets read, and b) it is much more likely to be maintained and up-to-date. Detailed design documentation is generally best produced by reverse-engineering models from the code as required. Business rules are generally much more stable than source code and hence business rule documentation should be more stable and maintained.

Enterprise Agile

Agility by Contract

An enterprise agile process retains the principles of an agile approach without discarding enterprise-scale process best practice. It addresses some of the shortcomings discussed above by formalising some of the key communication boundaries and scalability concerns with specific process artefacts or contracts (see Figure 2) – the *business contract*, the *system contract*, and the *portfolio contract*.

Business contract – communication between the project and the business does not rely solely on software releases. Specific business models are created which capture and formalise business rules and business processes. Verification with the business can then happen rapidly before software gets built, and is scalable, allowing multiple participants to see how all the rules fit together at different times and in different locations. Further, it can be relatively system independent, allowing multiple system development and integration projects to be based on a common business contract.

System contract – communication between the BA, Dev team and PM, relies on there being a clear definition of the system requirements and its functional definition. These elements are known as the system contract and are formalised using use case models combined with system information models and operation contract definitions.

This clarity allows iteration plans to be linked to the broader release plans, and development dependencies to be managed.

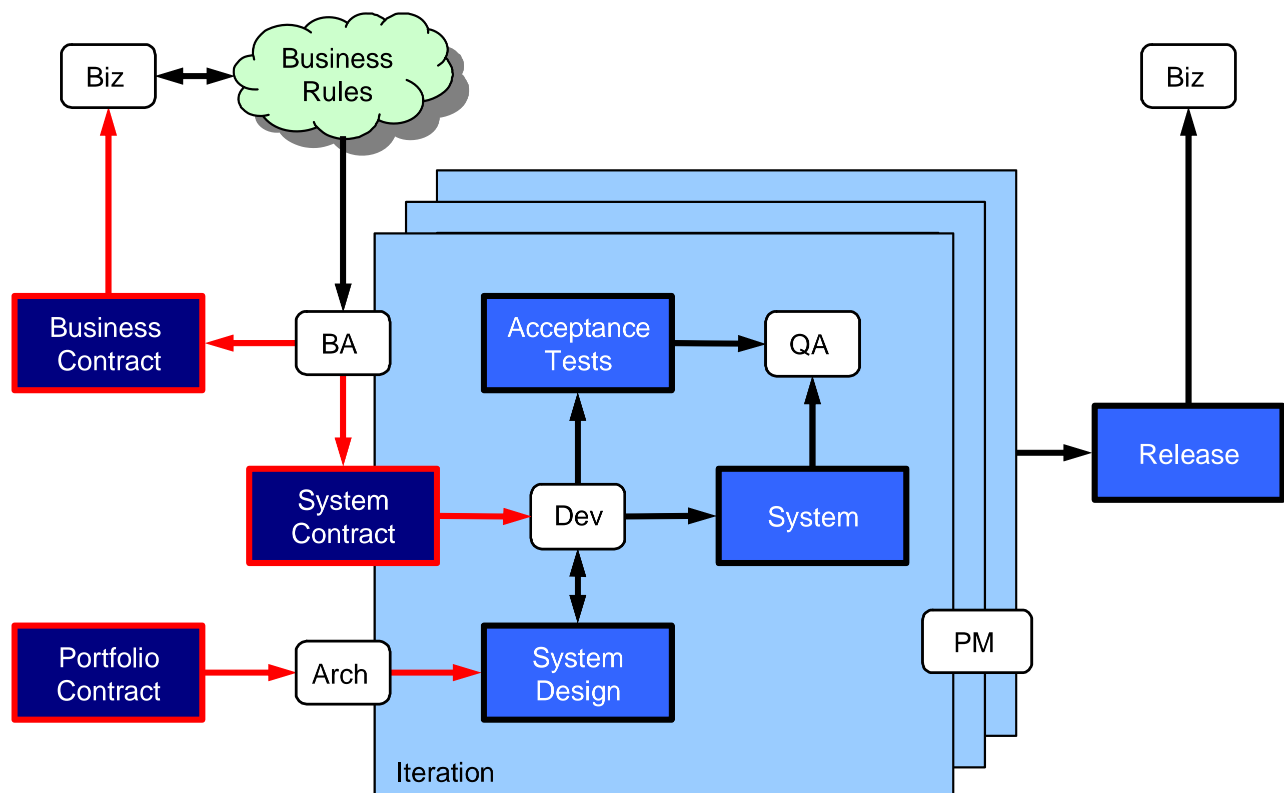


Figure 2 – Enterprise Agile Process

Copyright © 2006, Strata Software Limited

Further, by formalising the communication boundary between the business analyst and the development team, distributed development and off-shoring is better supported.

Portfolio contract – in an enterprise context system designs cannot be developed and managed in isolation. By establishing a clear link between an individual system design and the broader enterprise architecture, the role of architecture is enhanced allowing for portfolio-level re-factoring to be accommodated within projects focused on business functionality. The portfolio contract also further supports the construction of agile systems, distributed development and systems integration.

Incremental Contract Development

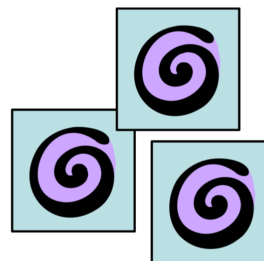
Formalisation of communication boundaries as contracts does *not* imply "big up front" contract development. By separating the business analysis and architecture workflow definitions from the iterative and incremental project management process, these contracts can be built up gradually as the project proceeds.

Each iteration establishes that subset of each contract as is necessary for software to be delivered. Of course, as some aspects of the contracts have a broader scope than an individual project, it may well be the case that the contracts already exist in full or partial form. The aim then is simply to standardise the form in which that information is captured.

Agile Development in a Box

In an enterprise context, design and implementation works very much as in the standard agile approach. It remains test-driven and iterative, it uses continuous integration to maintain quality, and it creates regular releases to validate requirements and priorities.

The main difference for the development workflow is that the boundaries and remit of certain practices are much clearer – releases are focused on analysis and design validation rather than business rule discovery; requirements are more precise so acceptance tests can be written more quickly; and the architectural boundaries are better-defined allowing re-factoring to occur with more confidence in a controlled environment.



Summary

Agile principles have caused everyone to look carefully at their current development practices. But agile methods were designed for small teams and small projects. This has meant that many organisations have attempted to run agile projects and failed because, although the principles are sound, current agile practices are not suited for enterprise-scale developments.

Enterprise Agile builds on existing agile principles but applies additional discipline, rigour and standards to allow agile practices to scale. It also acknowledges that projects do not exist in a vacuum, providing support for managing the business process context and system portfolio context on the project, without losing the delivery focus.

Enterprise agile combines the best of both worlds – the delivery focus of agile practices with a process framework that allows those practices to scale in an enterprise environment.

For further information please contact info@stratasoftware.com

or visit our website: www.stratasoftware.com

Copyright © 2006, Strata Software Limited