AGLPM1 – Unit 4 - ACTIVITY 2: OBSERVE

3rd Agile Leadership Skills: Encourage Incremental implementation

This document is an excerpt from the book:
"Agile Project Management for Government "
Authored by Brian Werham
Published by Maitland & Strong
Reproduced with permission under license

PMCAMPUS.com / Mokanova Inc

.

.

Agile Leadership Behavior Three: Be Incremental

> *Deliver (a working solution) frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
>
> Agile Manifesto Principle Three

In an agile project, transparency of progress is enabled by regular demonstration of parts of the working solution, early integration of different parts to make sure they work together, and incremental implementation at regular intervals. In the past, software projects have been the most obvious candidates for this approach because software can be constructed and dismantled quickly (compared to a building or aircraft development project, say). The trend now is for a wider set of projects to use agile approaches, such as business transformation programs and modular construction projects.

With the right approach, it becomes possible to identify any issues with the design before going too far. Then problems found in the previous iteration can be addressed, or at least placed onto the product backlog for fixing at an appropriate time without holding up the project. As the team plans the next iteration, they are aware of the practical feedback from the previous iteration and can avoid making the same mistake again. Iterative delivery at short intervals can be very expensive to do on many physical engineering or building projects. A waterfall approach to individual stages of a project may still be the best way forward (for example building a skyscraper). However, iterative delivery is often the key to success on many engineering projects.

Consider the success of the Apollo program. It succeeded in placing a man on the moon in just 8 years by carrying out progressively more challenging missions, one after the other. Initially, a waterfall approach was taken on the project.

A reluctance to change dangerous designs resulted in a disastrous cabin fire on the launchpad that tragically killed three on the Apollo 1 mission. That first mission was based on a design that was unproven and under tested. From that point onwards, the Apollo program became ruthlessly methodical about feeding back lessons learned from

each iteration. Feedback from engineering tests was readily incorporated into the technology for the rest of the program. Working iteratively and progressively towards their goal, on the 11th mission, man landed on the moon.

Rework is not Wasted Effort

Agile embraces the concept of improving a solution by re-working a previously delivered solution. Rework sounds wasteful, but agile embraces this as the concept of *refactoring*. There is no shame in correcting mistakes, as long as this happens in a timely fashion. The idea is to create an initial version so that progress can be made, but with a deliberate intent of not addressing all issues in one go. The concept of refactoring rests on two important premises.

First, that there is an expectation that each technical developer should always reserve some time in each iteration to tidy up any structural defects he/she finds in modules – even if the defect is in an area that is not planned for work in that iteration. Every module should be in a better structure at the end of an iteration than at the beginning. 'Tidying up' the technical solution should take place every time a module is changed. In this way, the solution improves and matures with each iteration, rather than its internals getting more tangled and difficult to work on.

Second, where a deliberate decision is made to leave a technical problem unresolved, or a structure not as neat and tidy as it should be, this is recognized as a *technical debt* and placed as an outstanding item on the product backlog. In this way, known problems are identified and tracked, not 'brushed under the carpet'. The resolution of technical debt can be prioritized and planned into further iterations depending on its importance to the stakeholders.[i]

Using this concept, refactoring is an expectation and regarded as desirable refinement, not regarded as a cost of the failure to deliver perfectly in the first place. Refactoring promises that the solution will not degenerate when it is extended and modified over time, but will become more polished and elegant, perform better, and be more reliable with each iteration. This can also be seen as *embedding maintainability* into the solution life cycle. Refactoring is a way of incorporating practical feedback into the overall architecture that has been used so far.

At a macro level, refactoring may introduce changes to the overall architecture. If refactoring is at a micro level, in other words a rework of the internals of a solution, the changes in many cases may not even be noticeable by users of the solution, but may improve flexibility for the future and reduce maintenance costs.

Risks exist in this approach, so the team needs to liaise carefully to avoid problems. For example, refactoring the internals of a technical solution that is already working can introduce *bugs* – and these bugs may not be detected by the standard

tests that were used on the last version. Informal, ad-hoc refactoring can set the whole team back if the correction of a bug in one module has an unintended impact on another module. It can also be tempting to go beyond the planned scope of refactoring, start to dig deeper and attempt a grand rework of the internal design of a technical solution.

Test driven development is an XP technique where every change, no matter how small, is tested before the next change is made. This ensures that changes are made one step at a time and that if a mistake is made, it is caught immediately, and the source of the error becomes easier to pin down. This continual retesting can consume a lot of time, and so batteries of pre-agreed test data and automated test tools are necessary to ease refactoring. The running of these tests needs to be automated as much as possible to encourage continual and consistent retesting. Risks can be managed by encouraging co-ordination between team members, and between teams.[ii]

Iterative working patterns help achieve consistency. Deviations from standards are quickly noticed. Different methods achieve this in different ways. The waterfall approach assumes pre-agreement of detailed theoretical standards by 'quality assurance' teams, and the coercive enforcement of them by independent quality inspectors. A very different approach is taken in XP, where quality standards are agreed by the team and enforced by a natural norming process when pair programming is used. If it is discovered that the quality standards need refinement, the team adopts a more refined approach and spreads the word in an organic way – at standup meetings and by pair working. This is very effective on smaller projects, but once the team grows above a dozen or so people, organic processes need help. The DSDM framework provides a role to help this: the *Technical Co-coordinator* whose responsibility is to ensure that parallel development teams on larger projects work in a consistent way so that the solution is coherent, and that quality is good. A System Architecture Definition (SAD) is drawn up before starting to build the solution. DSDM specifically relates this to the IT aspects of design, but there is no reason why the SAD should not cover any aspects of required technology – buildings, communications, machinery, and so on. And, like any other agile product, leadership is needed to ensure that standards stated in the SAD are refined iteratively using practical feedback. The SAD should be a 'just enough' document, not part of a detailed BDUF.

Hierarchy of 'Delivery'

A strength of the agile approach is in the way that it creates a spectrum of delivery. At the one end of that spectrum is the delivery of a non-working model, or mock-up of just one aspect of the solution. At the other end of the spectrum is the full-scale rollout of a tranche of the solution for operational use.

However, when using a specific agile method one must be careful to understand what is meant by ‘delivery’. Agilists tend to use the term ‘delivery’ as a catch-all term – anything on this spectrum is often termed ‘delivered’, even if it is just a successful team integration test. Defining this spectrum of deliverability is important to provide transparency, demonstrate quality and prove control. But it is only the ultimate delivery of useful, working product and its operational embedding that enables benefits to be realized.

Agile methods, then, often oversimplify the richness of this spectrum. Government cannot just ‘ship a product’ – it is responsible for delivering outcomes, not technology. The Scrum method is an example of this oversimplification. It only provides a binary definition of delivery:

- *Done* product – meets a previously jointly agreed *definition of done* or *quality acceptance criteria*
- *Done and potentially releasable product* – not just meeting a set of acceptance criteria, but also having the potential for immediate use and business benefit.

The scope of the Scrum method is very much constrained to the internal world of a technical development team. The planning and execution of the implementation of the ‘product’ is visualized as merely a matter of ‘release’ or ‘shipping’ the product for immediate use:

> “The purpose of each sprint is to deliver Increments of potentially shippable functionality that adhere to the Scrum Team’s current Definition of ‘Done.’ Development Teams deliver an Increment of product functionality every sprint. This Increment is useable, so a Product Owner may choose to immediately release it.” [iii]

Some commentators have suggested that because Scrum is most used in situations where planning ahead is difficult, it can be forgiven for ignoring release management planning and skating over the inherent complexity of implementation. But if we wish to use the method on large-scale government environments we must consider the issue of the definition of the delivery spectrum further. [iv]

DSDM provides a more useful delivery model. It comes from a project framework perspective that incorporates the mechanisms involved in rollout of solutions, not just their build. In the DSDM framework, iterations target not two, but three levels of *deliverability*:

- Purely *Exploration*, where the stakeholders’ requirements are explored through modeling and specification prototyping
- A mix of *Exploration* and *Engineering,* where an interactive development takes place to create a working solution which can be demonstrated

- *Deployment*, where the working solution is implemented and business benefit is realized.

Keith Richards suggests that this, among other reasons, explains why, in his opinion:

> "Only DSDM can be used 'as is' for projects. Scrum and XP are product delivery techniques – they have no concept of 'a project'." [v]

But these methods do, to an extent, oversimplify the planning needed to meet Agile Manifesto Principle Three to deliver a mutually recognizable working solution on a regular basis. Therefore, it is instructive to examine a more differentiated type of hierarchy of delivery. The military defines nine Technology Readiness Levels (TRLs) to categorize delivery: [vi]

1. Basic – the lowest level of technology readiness where theory awaits practical implementation
2. Technology concept formulated – some basic principles demonstrated, but still essentially theoretical
3. *Proof-of-concept* – some physical validation of parts of the design
4. Subsystem/Component validation – laboratory environment integration of whole parts of the system
5. Subsystem/Component validation in a relevant environment – technological components tested in a simulated environment
6. System prototype – relevant demonstration – a simulated operational environment demonstration
7. System prototype – operational demonstration – the demonstration of an actual system prototype in an operational environment
8. System completed – technology qualified through test and demonstration in expected conditions
9. Successful mission operations – application of the technology under mission conditions.

Table 1 introduces the delivery-planning concept, which will help government agile projects demonstrate mutually agreed regular deliveries.

Level 1 Delivery – Specification Prototyping

Level 1 deliveries are useful to elicit feedback and gain buy-in and the confidence of stakeholders – especially at the start of the project. The DSDM framework recommends that a *feasibility prototype* should be used while an *outline business case* is created, but before the feasibility of the project overall is confirmed. This work will by nature be limited and should focus on the business issues at hand. The objective is not to start building a solution, but to investigate any practical aspects of the solution that could impinge upon financial costs/benefits, and to gain confidence before a full-scale business case is created.

Level 2 Delivery – Demonstration of an Emerging Solution

Level 2 deliveries are not intended for use, but give realistic assurance to management and stakeholders about critical aspects of the solution:

- Is the solution 'user friendly'? Is the citizen uptake of the new service that is presumed in the business case still realistic?
- Does the solution perform? A set of what are termed 'non-functional requirements' will need to be developed and tested to ensure that performance is adequate.
- Is the internal architecture developable and maintainable? It is difficult to ensure that complex solutions will work as intended.

Table 1: Proposed Hierarchy of delivery for Agile Projects

Level	Deliverable	Description
1	Specification Prototype	A working model of some aspects of the solution.
2	Emerging Solution	A partially constructed solution that conforms to technical standards, but still has functions missing required for real-world use.
3	Shippable	A partially constructed solution that could be used.
4	Consumable	A partially constructed solution that is ready for use.
5	Piloted	An increment of the solution deployed and in use in a limited locality, customer segment and/or for a limited time
6	Implemented	An increment of the solution that is in wide-scale rollout/use.

Level 3 Delivery – Having a 'Shippable' Product Ready

Level 3 delivery is simply the 'potential' for delivery. Periodically, a whole solution is built and proven to work – in theory. The stakeholders may not be ready to start using the solution, and other implementation restrictions may exist. For example, the UK Department of Work and Pensions has approximately 10,000 staff working in unemployment offices. Changes to the computer systems must be carefully made in conjunction with staff training, and it is not thought practical to make wholesale changes to processes on a regular basis. Level 3 delivery has the following advantages:

- Further feedback can be gathered from stakeholders on the requirements and their priority
- Confidence in the progress of the team is increased
- The technical infrastructure of the solution can be proven end-to-end, thus reducing the problems of integration of many different components.

However, the delivery of a solution by the development team at Level 3 does not guard against future problems such as:

- Whether the solution can be easily implemented and used for full-scale operations

- Whether the preparation and training planned for users of the technology is adequate
- Whether the user acceptance tests are adequate and have the coverage required for real-world use.

Level 4 Delivery – Having a 'Consumable' Product Ready

This delivery level requires user acceptance of the solution, and the delivery mechanisms tested. The delivery is not just 'shippable' from the point of view of the developers, but also 'consumable' from the point of view of the stakeholders.

Setting up delivery mechanisms to make implementation smooth requires activities that Carl Kessler calls 'meta-tasks' – in other words additional planning and development that is necessary not to build a solution, but to build the processes required so that the solution can be implemented.

Dry runs of the product on real-life data may take place to show that the new technical solution is usable. Where business rules are unchanged, parallel running can help identify any errors in the running of the new solution.[vii]

Level 4 delivery incorporates Kessler's concept of 'consumability': a product needs to be more than just 'shippable', but also 'consumable'. He advises a three-step process to ensure this:

- Identify 'consumability meta-tasks': those activities that need to be carried out to smooth the path to a successful use of the solution
- Treat consumability capabilities like any other solution capability: implementation and use needs to be formally tested
- Set up measures for 'consumability' and continually improve the solution in this respect. [viii]

Level 5 Delivery – Piloting the Solution

This level requires pilot running of the solution. The UK Green Book states that:

> "(One benefit of a pilot is that it should) acquire more information about risks affecting a project through (and allow) steps to be taken to mitigate either the adverse consequences of bad outcomes, or increase the benefits of good outcomes." [ix]

The GAO has recently audited pilot projects as diverse as tax collection at the Internal

Revenue Service (IRS) and implementation of improved emergency planning at the Federal Emergency Management Agency (FEMA), and has developed criteria for successful planning for pilot projects. These are:

- ♦ Well-defined, clear, and measurable objectives
- ♦ Criteria for determining pilot-program performance
- ♦ A method for the determination of appropriate pilot size and a strategy for comparing the pilot results with other efforts
- ♦ Plans for data collection and analysis to track the program's performance and evaluate the final results of the project. [x]

Of course, usefulness of pilots goes beyond providing risk management and providing technical feedback to the solution developers. Pilot implementation of a solution can engender transparency because significant stakeholders will be involved in checking that the new way of working really does bring business benefits. Plans for the best approach for rollout of a solution and its speed will be informed by its pilot usage.

Level 6 Delivery – Widespread Phased Rollout

Level 6 is real-world delivery, free of the constraints of the theoretical testing of levels 1 to 4, and the limitations of piloting. The costs and benefits of wide-scale rollout and usage can be measured, and the business case tracked. Level 6 delivery should be incremental. An example of an incremental implementation approach was that taken by the US Department of Veterans Affairs (VA), where since 2009 projects are required to have processes to build and deliver incremental functionality every 6 months. A 'three strikes and you're out' policy repeated failure to deliver functionality as scheduled will result in a project being paused – or even terminated.[xi] The avoidance of a big-bang implementation gives opportunity for early identification of project problems. Carefully used it can guard against the 'scope creep' that can often occur on projects where new features and requirements slowly are added to an overlarge requirements catalogue.

Conclusions

Delivery of a partially working solution provides transparency on the progress of a project – especially if an agile approach is taken which delivers incrementally, and in very short iterations. This will allow for incremental improvement of the solution by refactoring the design and architecture.

You may notice that Agile Manifesto Principle One requires "continuous delivery",

whereas Principle 3 refers to “short iterations”. One should not be too literal in comparing these principles. Principle 1 reflects the ideas behind *lean* development techniques discussed in Part 1. Lean development is almost indistinguishable from *continuous improvement* approaches used in running normal day-to-day operations.[xii] *Kanban* is a lean technique that involves displaying status cards on walls to illustrate the planning of outstanding work and progress against schedule. It is simply the Japanese word for “visual information”. The idea is that as soon as something changes, whether it be a new issue that is identified, or an existing task that is re-prioritized, the cards are changed on the wall, and everybody nearby can see the change immediately.

When scaling up agile techniques for large-scale government projects a decision must be made as to what ‘continuous’ delivery means, and that requires two factors to be agreed for each iteration. First, the criteria for acceptable delivery. Second, the length of time to be taken. Near continuous delivery by very short iterations is most suitable where operations can respond nimbly to change. Longer iterations are more suitable where the nature of operations require periodic releases at longer intervals, or where the build times for the technology are measured in months, rather than days. The important thing is to do this with eyes wide open, and Agile Leadership Behavior Two requires a presumption towards a shorter length, rather than longer wherever practicable.

VA has required incremental implementation since 2009. Their ‘three strikes and you’re out’ policy ensures that project delivery happens regularly. If more than two incremental deliveries are missed, then the project is stopped and considered for major replanning of approach or cancellation.

Agile methods, such as DSDM and Scrum, recognize that ‘delivery’ does not always imply actual usage. The creation, testing, and acceptance of increments of work results in a type of delivery. It is objective and auditable. But it is not always practical to put every changed piece of software live right now. Therefore there is a risk of a lack of nuance over what ‘delivery’ means, and (especially in Scrum) what the development team’s responsibility for implementation are.

Implementation is, for most government projects, much more than just ‘shipping the product’. For this reason (and others which I will explore later) some agile experts argue that DSDM should be used as a ‘wrapper’ around delivery techniques such as Scrum and XP which are focused more at team level than corporate level. A useful planning concept is for the development team to target a ‘hierarchy of delivery’ so that the type of delivery from each increment of work is clear and expectations are set with stakeholders, especially with those tasked with training and implementation of the solution with end-users.

[i] {Fowler 1999 #187}
[ii] {Fowler 1999 #187: Gamma, Eric in foreword}
[iii] {Schwaber July 2011 #118: 15}
[iv] {Dybaa 2008 #93: 3}
[v] {Richards 2010 #329}
[vi] {Levin 2003 #341: Enclosure II, Table 2}
[vii] {Kessler 2007 #191:72}
[viii] {Kessler 2007 #191:72}
[ix] {Great Britain. Treasury 2003 #182: 81}
[x] {U.S. Government Accountability Office 07/11/2008 #192} and {U.S. Government Accountability Office 08/04/2011 #193}
[xi] {VA Office of Inspector General 25/08/2011 #194}
[xii] {Williams 2012 #239}