Hybrid-Agile Software Development

Anti-Patterns, Risks, and Recommendations

Paul E. McMahon, PEM Systems

Abstract. Many organizations are driving toward increased agility in their software development practices. However, due to various constraints (e.g. project size, team physical distribution, compliance requirements, technical complexity) a pure Agile approach is not always feasible. This leads to what today is commonly referred to as a "hybrid-agile" [1, 2] approach. Using a hybrid-agile development approach requires that organizations think carefully about process tailoring and metrics decisions to ensure they stay aligned with their performance goals.

The purpose of this paper is to provide motivation for hybrid-agile approaches, identify common challenges hybrid-agile projects face today, and to provide recommendations that can help teams using a hybrid-agile approach reason through their challenges leading to more effective process tailoring and metrics decisions. Anti-patterns and related risks commonly observed today on large complex hybrid-agile efforts are identified and employed as an aid in demonstrating the reasoning process.

Introduction

The Goal - Question – Metric (GQM) approach to determine optimum metrics [3] has long been accepted as the gold standard for metrics identification. With GQM you start by asking:

"What is our goal?"

Then you formulate a set of questions that can help you assess how well you are doing toward achieve the goal. This leads to a set of metrics to collect that will help answer those questions.

What is the goal of measurement on most software development projects?

First, most organizations measure to understand where they are with respect to where they planned to be so that corrective action can be taken when necessary. An example could be to add resources when your measurements indicate you are behind schedule.

A second reason many organizations take measurements is to help improve performance. These improvements could be on the next project, or the next iteration of the current project.

Understanding your goal is important because the best measurements to collect in a given situation depend on what you are trying to achieve. Let's now look closer at what organizations are trying to achieve when using a hybrid-agile approach.

Why Hybrid-Agile?

In today's rapidly changing, competitive, fast- paced world organizations need to be able to get product to market faster, and they need to be able to respond rapidly with product changes to address changing customer needs. However, many of these same organizations also live in highly regulated environments where compliance to standards is equally critical to business success.

Part of being effective in responding to change and complying with regulations is being able to predict how long all the work will take to get new or modified features ready for stakeholder use while ensuring no critical steps are bypassed.

To predict we must be able to estimate the work effort, but unfortunately accurate software cost/schedule work estimation has alluded the software community since the early days of software development. This observation is not new. Over 15 years ago Tom Demarco and Tim Lister explained the problem as follows [4]:

"Most software managers do a reasonable job of predicting the tasks that have to be done and a poor job of predicting the tasks that might have to be done."

While the problem of predicting the tasks that might have to be done has been difficult for the software community for a long time-- and especially difficult on large complex software efforts-- the recent agile movement has given us some new ways to think about, predict, and measure progress.

What is Different in How Agile Projects Measure Progress?

A major reason why the agile movement continues to gain steam even after ten years is the recognition that many stakeholders do not fully understand the requirements for their desired software system at the start of their endeavor. Furthermore, most professionals involved in software development today know we need a better way to deal with rapidly changing requirements even late in the project.

Agile practices emphasize the need for development teams to work closely with stakeholder representatives to uncover the stakeholder's real needs and manage the resultant work from the start of the endeavor through its completion.

Nevertheless, as organizations have tried to implement these promising new agile practices-- particularly in large, constrained environments-- the resultant agile-tailoring's have led to difficulties and a number of commonly observed anti-patterns.

Anti-Patterns, Observations, Risks and Recommendations

In this section seven anti-patterns commonly observed on hybrid-agile efforts are identified and discussed. For each anti-pattern, observations, risks and recommendations are provided demonstrating a reasoning process[1] [6] that could help organizations using a hybrid-agile approach make better process tailoring and metrics decisions.

Anti-Pattern One: *Creating an "aggregated" team velocity metric and using it to predict and drive progress*

Observations, Risks and Recommendations

Team velocity is a metric that is used to measure the amount of work an agile team estimates it can complete in the next Sprint[2] [7] based on recent team performance.

Most organizations looking to increase agility today understand the importance of creating small Scrum[4] teams even on large complex efforts [8], but many don't yet understand the importance of empowering these small teams with the responsibility to measure themselves.

Often what we see on large hybrid-agile projects is the velocity being set at a high level in the organization and given to the small teams, rather than allowing the small teams to measure their own velocity and then set the target work for the next iteration based on their own recent past performance.

The most accurate estimates of effort to complete work are based on recent performance of each specific small team doing the work. Each small team should measure its own velocity. Organizations should anticipate varying team velocities for each small team.

Defining team velocity from the top defeats the purpose of the velocity metric. When used appropriately Scrum teams should get better at predicting the work that can be accomplished in each sprint as they progress from sprint to sprint learning from their own velocity.

It is recommended that you let your project manager know where he or she can see each small team's velocity measurements in a place where it is visible to the whole team, such as on the wall in a room where each small team holds their daily standup meetings[5] [7]. This will keep the true velocity visible to the team and will reduce the temptation for intermediate and senior managers to use this metric inappropriately.

Anti-Pattern Two: *Telling the team to work harder to improve performance.*

Observations, Risks and Recommendations

When progress is not being achieved per the plan too often the response has been to tell the team members to work harder, such as by putting in overtime hours, to improve performance. While overtime can help to improve performance in isolated instances, excessive regular overtime risks team burn out, and it does not get to the underlying root cause. Furthermore, in these situations, team members often respond by cutting corners and not following their agreed to way of working (e.g. reducing their planned testing or peer reviews). Ultimately this leads to more latent defects and longer schedules.

One approach that can help is to encourage your teams to use the story point efficiency metric. Story point efficiency [6] is defined to be the ratio of the estimated time to complete a user story[6] [9] divided by the actual time it took. This metric can help teams quickly identify problem areas in their requirements/user stories and investigate those problems in a timely way to learn what is hindering the team from achieving their estimate. It is recommended that you keep each small team's story point efficiency measures visible to the whole team, and encourage them to use this measurement data to continually improve their velocity.

When teams don't hit their estimates it usually isn't hard for them to figure out why, if you give them the time to investigate the situation right when the problem is happening. This is also the best time to resolve the problem because it doesn't delay the resolution to the next project, or even the next iteration of the current project. It can help the team's performance right when the problem is happening by raising the visibility of the problem to the right level and getting it resolved in a timely way.

Agile teams must own their practices and their continual improvement

A key strength teams gain from using the story point efficiency metric is that it gives teams the data they need to put improvements in place. This includes identifying weaknesses that have slowed them down in the past and putting improvements, such as better checklists, in place to catch similar potential problems in the future. If they still don't see velocity improvements after implementing changes, it is likely they are not putting the right improvements in place to resolve the problem. Agile teams are self-directed which means it is the responsibility of the team members to identify and resolve performance issues.

Agile practices can work only if the agreed way of working within the organization empowers the team to make timely changes necessary to improve without going through bureaucratic approvals outside the team.

Anti-Pattern Three: *Failure to actively manage risk exposure at the small team level.*

Observations, Risks and Recommendations

Today on many large complex efforts we see risk management carried out at the senior management level, and not effectively implemented deep into the organization. A key agile principle is to take on risky work early driving overall project risk down as the project proceeds [10]. Risks should be actively identified and managed at the small team level, and then rolled up consistently to the higher level—not the other way around. When small agile teams plan their work for each sprint they actively discuss risks to ensure they are driving the risks down early.

It is recommended that you keep the risk trend visible to the team to help the team discuss the right issues when making key work related decisions for the next sprint. The risk assessments at the small team level should be rolled up in a consistent way to provide an accurate overall project risk assessment.

Anti-Pattern Four: *Failure to actively manage stakeholder involvement and stakeholder representation competency.*

Observations, Risks and Recommendations

Too often we see organizations trying to increase their agility at the development team level, but failing to recognize that agility requires changes in the behavior of the stakeholder community as well. To gain the benefit of agility stakeholder representatives with the right competencies must be assigned, agree to their responsibilities, and be given the time to carry out their responsibilities in a timely fashion.

Too often we see the stakeholder representatives that are assigned are people with inadequate competency to carry out this critical role. The stakeholder representation role requires people with the ability to gather, communicate and balance the needs of other stakeholders, and accurately represent their views—not just their own views. [6]

Anti-Pattern Five: *Using "proxy releases" rather than formally selling-off products incrementally.*

Observations, Risks and Recommendations

On large projects, due to various constraints, it is often the case that products cannot be made operational every sprint. Therefore large projects typically conduct multiple sprints leading to each release. On very large projects a release could be every 6 Sprints (or every 6 months). However, these releases should be real "sell-offs" of product to the real stakeholders/customer.

An anti-pattern commonly observed is for these releases to be conducted as "proxy releases." By "proxy release" I mean a release which is not a formal/official sell-off to the real stakeholders. Rather it comprises some level of testing and demonstration in the presence of someone representing the stakeholders, but not authorize to formally accept the product.

The rationale provided for "proxy releases" often relates to constraints such as inability to operationally deploy partial functionality, and/or the lack of availability of key stakeholder representatives. However, the fact that the product cannot be deployed on short cycles should not get in the way of stakeholders being involved, responsible, and given the time to collaborate and accept functionality on short cycles.

The risk with "proxy releases" is that too often, when a real product acceptance is not conducted, we see the development teams and stakeholder representatives just "going-through-the-motions" and not rigorously testing against the agreed to requirements/features allocated to that release.

Often these "proxy" events do not have stakeholder representatives that are authorized, and knowledgeable to provide real answers and to conduct a thorough review of the product. Ultimately this leads us back to the traditional integration and test problems late in the project and the goal of getting product and product changes to the customer rapidly is not achieved.

I have probably heard most, if not all, of the reasons why "we can't" sell-off incrementally.

Yes, it takes authorized and knowledgeable stakeholder representatives from the customer side.

Yes, the team needs to complete all their work that they have committed to following their agreed way of working including thorough testing.

Yes, all the key stakeholders need to agree that the software system is worth making operational.

But these are the points why agile works. A key agile practice is to get the work done in short increments and get product to customer sooner which also reduces the risk of late surprises, extended schedules, and cost over-runs.

Why is this so important?

If you aren't measuring features accepted by the customer incrementally, you haven't really understood why agile development can help you achieve your ultimate goal, and you probably won't.

Anti-Pattern Six: *Essentially traditional development with a few "agile practices" sprinkled in to make the project appear "agile."*

Observations, Risks and Recommendations

I have observed many large projects that claim to be using agile methods to be essentially traditional development with a few agile practices being conducted by the development teams. When I have looked close at these projects that are essentially traditional I often have found a "business as usual" attitude even though they may use the "agile" buzzword. The risk in this approach is that it is highly unlikely they will ever experience the real potential value of agility.

One way to counter this risk is to use the "how agile are we?" metric. The "how agile are we?" metric gives you an indication of the degree of agile practice adoption by your team. There are numerous ways to measure how agile you are, many of them by survey. Examples include the Shodan Adherence Survey and the ComparativeAgility Assessment [11].

For many years I did not like the "how agile are we?" metric because my view was it didn't matter how agile an organization

How Have We Traditionally Measured Progress?

Traditional approaches to measure progress are well known along with their shortcomings. The most widely used approach on large complex software efforts is Earned Value Management (EVMS) [5]. The fundamentals of EVMS include breaking the work down into small pieces, estimating the cost of each piece, and monitoring actual expenditures against an agreed to baseline plan.

One weakness with EVMS rests in the assumption that we know all the work that must be done when we plan it, and we know how long each piece will take to complete. Another weakness is the fact that even if we get all the identifiable work done it doesn't necessarily mean the stakeholders will be satisfied that the software system is ready for use.

Traditionally many organizations that use EVMS have addressed these weaknesses by applying risk management practices. However, the way these practices have been carried out in many organizations have failed to adequately address these weaknesses in a timely way.

For example, in one of my client organizations I heard that when a potential risk is raised so much collateral evidence had to be gathered that effectively they had to prove the risk was already a problem before the risk board would accept it.

In another client organization I heard that no one ever raises a schedule risk because the culture in the organization had become one that just accepted the fact that all good schedules had to be aggressive and risky given today's business climate. So risk in a schedule was no longer perceived as a risk that needed to be managed. Rather it was now perceived as an acceptable part of the normal way of working due to the aggressive business climate.

was, it was more important that they had the "right level of agility" given their situation. But I have since discovered that the "how agile are we?" metric can give an organization an early indication of the likelihood that their "brand-of-agile" will help them achieve their performance goals. This metric can give you a good idea early in your project of the level of commitment your organization really has to agile practices.

A Simple Way to Measure "How Agile Are We?"

While hybrid-agile projects can vary across a continuum they could be characterized at the extremes and in the middle through a simple three level scale as follows:

Case 1: Fully Committed

Fully committed means the project has adopted an agile approach across the entire program including systems engineering, test, and stakeholder participation. Characteristics of the fully committed case include:

- Requirements expressed in user story form in a backlog
- The backlog provides the one and only list of requirements
- The backlog is refined, and reprioritized at the start of each sprint
- Systems engineering and test is integrated into the small scrum teams
- Authorized stakeholders provide product owner role attending sprint demos accepting product deliverables at sprint level

Case 2: Hybrid Agile/Traditional

There could exist varying levels within this case, but the typical characteristics include:

- High level requirements completed early in project by systems engineering and allocated to multiple sprint releases
- Lower level requirements generated as user stories and managed by small scrum teams through backlog
- Some level of stakeholder representative involvement at sprint demos
- Multiple sprints lead to release sell-off at sprint release level with authorized stakeholder representatives present

Case 3: Essentially traditional program with a few agile practices conducted by the software teams.

Characteristics of this case include:

- Requirements developed up front by systems engineering
- Requirements developed and managed traditionally in tool such as DOORS[7]
- Software scrum teams break high level requirements down into user stories and manage through backlog within sprints
- Multiple sprints lead to release sprint
- Sprint demos may be conducted to get early feedback at release sprint, but no or minimal acceptance/sell-off of product at sprint release level
- All or majority of requirements tested at end of project through traditional integration and test/acceptance

Risks Associated with the Three "How agile are we?" Hybrid-Agile Cases

Case 1: Requires significant investment by customer to train and commit customer personnel to participate regularly in project activities throughout lifecycle.

Case 2: When choosing a hybrid agile/traditional approach, if personnel involved lack agile experience, there is risk of poor tailoring decisions (e.g. practices, metrics) leading to failure to achieve the intended agile benefits.

Case 3: A major value in using agile approaches is to improve contractor-stakeholder communication and reduce the risk of unexpected latent defects and cost/schedule overruns. This value is unlikely to be achieved in Case 3 since the authorized and knowledgeable stakeholder representatives are not engaged throughout the endeavor.

Anti-Pattern Seven: *Using the requirements volatility measure inappropriately to control scope creep.*

Observations, Risks and Recommendations

Requirements volatility is a common metric that has been used traditionally to manage requirements scope creep. Often this metric continues to be used in an inappropriate way on hybrid-agile endeavors. When using agile practices trying to control project cost and schedule by minimizing requirements changes can conflict with recognized best agile practices. With agile practices you collaborate with your customer to provide the best value for the available resources. Therefore as the project proceeds it may be fine for changes to occur in priority and content of the requirements backlog. When you move to an agile approach continual and close collaboration with the customer trumps controlling requirements volatility.

If you have an effective collaborative relationship with your customer it can be beneficial to allow requirements volatility (e.g. requirements changes late). Requirements stability isn't the end goal. Stakeholder satisfaction is.

Summary

The purpose of this paper has been to provide motivation for hybrid-agile approaches, identify common challenges hybrid-agile projects face today, and to provide recommendations that can help teams using a hybrid-agile approach reason through their challenges leading to more effective process tailoring and metrics decisions. Anti-patterns and related risks commonly observed today on large complex hybrid-agile efforts were also identified and employed as an aid in demonstrating the reasoning process.

ABOUT THE AUTHOR

Paul E. McMahon, Principal, PEM Systems (www.pemsystems.com) has been an independent consultant since 1997. He has published more than 45 articles and multiple books including "15 Fundamentals for Higher Performance in Software Development." Paul is a Certified Scrum Master and a Certified Lean Six Sigma Black Belt. His insights reflect 24 years of industry experience, and 17 years of consulting/coaching experience. Paul has been a leader in the SEMAT initiative since 2010.

E-mail: pemcmahon@acm.org

NOTES

1. The Essence framework was used as a guide in developing the referenced "reasoning process"
2. The heart of Scrum is a Sprint. A time-box of one month or less during which a "done", usable, and
potentially releasable increment of software is created.
3. Experience has proven that effective Scrum teams should be no larger than 8-10 people
4. Scrum is a framework for developing and sustaining complex products.
5. A daily standup meeting, also referred to as a daily Scrum, is a time-boxed 15 minute meeting each day where the team synchronizes activities for the next 24 hour day.
6. User stories is one way to describe requirements when using agile software development
7. DOORS is a commercially available requirements management tool

REFERENCES

1. Ambler, Scott, Lines, Mark, Disciplined Agile Delivery, IBM Press, 2012
2. McMahon, Paul E., "Integrating CMMI and Agile Development: Case Studies and Proven Techniques for Faster Performance Improvement, Addison-Wesley, 2011
3. Humphrey, Watts, "A Discipline for Software Engineering", Addison-Wesley, 1995
4. Demarco, Tom, Lister, Timothy, "Waltzing with Bears", Dorset House Publishing, 1995
5. Solomon, Paul J., "Practical Performance-Based Earned Value", Crosstalk, The Journal of Defense Software Engineering, May, 2006
6. McMahon, Paul E., "15 Fundamentals for Higher Performance in Software Development", PEM Systems, http://amzn.com/099045083X. July, 2014
7. Sutherland, Jeff, Schwaber, Ken, "Scrum Guide-- The Definitive Guide to Scrum: The Rules of the Game", July 2013
8. McMahon, Paul E., "Lessons Learned Using Agile Methods on Large Defense Contracts," Crosstalk, The Journal of Defense Software Engineering, May, 2006
9. Cohn, Mike, "User Stories Applied: For Agile Software Development, Addison-Wesley, 2004
10. McMahon, Paul E., "Defense Acquisition Performance: Could Some Agility Help?", Crosstalk, Journal of Defense Software Engineering, February, 2009
11. Cohn, Mike, "Succeeding with Agile: Software Development Using Scrum, Addison-Wesley, 2009