**Introduction**

Nothing has ever been invented — not the light bulb, not the automobile, not the iPhone. Someone at some point got the notion to combine two or more existing ideas and call it the next new thing.

So it was with the Agile Alliance, the Agile Manifesto and its nine principles. In February 2001, 17 “anarchists” met to create a common and more efficient method of software development. They determined what was valuable, productive, essential, negotiable, deliverable, trustworthy, sustainable, creative and simple. [1] Eight of the “anarchists” then posed for that famous brown tinted faded photo as though they had just drafted the Declaration of Independence. But as with all inventions, their work was based on the ideas of at least two famous individuals at least a century or two earlier.

In the summer of 1855, 16-year-old John D. Rockefeller, the itinerant son of a snake oil salesman, unable to afford a formal college education, completed a three-month course of study at E. G. Folsom’s Commercial College, a chain trade school with branches in seven cities. He acquired working knowledge in double-entry bookkeeping, penmanship, and the essentials of banking, exchange and commercial law. By January 1857 he had been elevated to chief bookkeeper for a Cleveland merchant, mainly because his limited business training allowed him to determine whether his company was gaining or losing money and to enact process improvements to make more money (unlike his competitors, who generally had no idea they were losing money until they went out of business).

Shortly thereafter he started “Clark and Rockefeller,” which bought and sold carloads of produce. For him, the Civil War was principally an opportunity to pile up riches. Once oil was struck in western Pennsylvania, his company took on consignment some of the first crude oil shipments, which reached his Cleveland refineries in early 1860. Known today erroneously as an oil baron, John D. was known as the “Kerosene King,” owning multiple refineries and controlling the trains and pipelines that carried the illuminant around the world. It would be 50 years before John D. found a useful purpose for gasoline, an unwanted byproduct of the oil refinery process. [2]

John D. was one of the first to apply the nine Agile principles. Before he began distributing kerosene, only rich people could light their houses after dark (candles cost money). He satisfied customers through early and constant kerosene delivery (OK — he broke a few laws making deals with the railroads, but that forced him to satisfy the Agile concept of reducing documentation). He welcomed changing requirements by going from produce to oil and shortened timescale by making (mostly illegal) deals with suppliers face to face. He had to work with the railroad and pipeline suppliers, motivating their self-organizing teams through greed. Whenever government regulators questioned his business acumen, he would reflect, tune and adjust his behavior to satisfy their concerns.

A century earlier, Alexander Hamilton also began as a clerk at 16. He worked for the firm Beckman and Kruger in St. Croix. However, a grim tally of disasters preceded this. Between 1765 and 1769, his father vanished, his mother died, his cousin and supposed protector committed bloody suicide, and his aunt, uncle and grandmother all died. Alexander was left alone, friendless and penniless, surrounded by failed, broken, embittered people. His short life had been shadowed by a stupefying sequence of bankruptcies, marital separations, deaths, scandals and disinheritance. Such repeated shocks must have stripped Hamilton of any sense that life was fair, that he existed in a benign universe, or that he could ever count on help from anyone. But help finally arrived in the form of benefactors who sent him to America. There he attended college for a time, then became George Washington’s adjutant during the Revolutionary War. After that, he became Secretary of the Treasury under Washington’s presidency. [3]

Hamilton worked together in the field with Washington, coordinating self-organizing commands through face-to-face communications, motivating soldiers to work together to maintain a constant pace of battlefield victories on a short timescale. When faced with adversity, he and Washington reflected, tuned, and adapted their behavior until their eventual victory at Yorktown. As Treasury Secretary, Hamilton satisfied his mercantile customers with early delivery of the First National Bank (to the chagrin of his agrarian enemies Thomas Jefferson, James Madison and James Monroe, proving you can’t please everybody). When his plan to fund the government through customs duties was threatened by smugglers, he quickly changed requirements and created a customs enforcement navy that eventually became the Coast Guard.

**So You Want to be an Agile Project Manager**

A lot has changed in the past couple centuries, but those basic Agile concepts are still valid. A typical Agile team consists of a product owner (PO), Scrum master (SM), and team members...
who perform multiple tasks (defining scope, establishing require-
ments, designing, coding and testing). Notice I didn’t list project
manager (PM), since most Agile teams do not have one. Hamilton
and Rockefeller would likely fall into the PM/PO category. In a
recent unscientific blog post[4] that I posted (there is very little
research in this area), I received 72 comments on this subject
from both Agile PMs and Agile practitioners. Most PMs thought
that their skills could benefit Agile teams. However, most Agile
practitioners not only viewed the PM as evil but also considered
all project managers to be wastes, bottlenecks and a hindrance
to velocity. Many questioned whether a PM should be allowed to
breathe oxygen in an Agile environment. The typical PM is consid-
ered to be like a wart on a frog: defined in the SOW as necessary
(a holdover from Waterfall), but no one is quite sure what he or
she does. I’ve personally met several Agile PMs who admitted they
really had very little to do. I believe this is because while there are
many books, tutorials and classes that teach project management
and Agile methodology, no one has ever attempted to define Agile
PMs in a way that warrants respect and usefulness.

So let’s try to define some major life cycle processes — and
I’ve purposely chosen some that make most Agile practitioners
cringe — where the PM can demonstrate effectiveness: require-
ments management, Agile leadership, configuration manage-
ment and change control, team composition and management,
scope and estimation, monitoring and control, measurement and
analysis, Agile product integration, Agile release planning, and
prioritization and tracking. Because many Agile PMs are their
own worst enemy, as Hamilton and Rockefeller often were, fail-
ing to understand their basic role in an Agile environment, let’s
start with requirements management and leadership.

Requirements Management

“And you want to be my latex salesman?”

—Jerry Seinfeld, “The Boyfriend” episode

Since Scrum is the most popular of the Agile frameworks,
this article will target process challenges for that method. It
should be noted that multiple frameworks can be combined
(e.g., Kanban can be used within Scrum to control team capac-
ity constraints by limiting work in progress). While the goal
for management is to make sure the product/iteration backlog gets
fed and can support and describe what needs to be built by the
development team prior to the start of the sprint, many project
managers and organizations struggle to understand value-based
decomposition and backlog prioritization. [5]

Hamilton spent a great deal of time convincing the states
to ratify the Constitution over the existing Articles of Confed-
eration. The first would create a powerful centralized federal
government while the latter provided more states’ rights, so the
South figured, “Why fix what ain’t broke?” As Hamilton negoti-
ated with each state, requirements changed (Virginia demanded
slavery; New York was abolitionist; South Carolina was agrarian; the
North mercantile). Similarly, management of Agile require-
ments differs from traditional requirements management be-
cause requirements evolve with iterations. With Agile methods,
the team does not attempt to specify fully detailed requirements
up front. Instead, the team initially establishes “course grained”
requirements and then progressively elaborates on them as
the process progresses. The requirements are refined from the
product backlog into the iteration goal, then further refined into
the iteration plan and then into user stories. The team continues
to refine the user story requirements during the daily stand-up
meetings and through other discussions. These are practi-
cal examples of how project elements get decomposed and
prioritized at the last responsible moment. The term “minimally
marketable feature” (MMF) is an important factor in a success-
ful iteration in order to avoid what is termed “gold plating” in tra-
ditional development environments. MMF refers to a package
of functionality that is complete enough to be useful yet too small
to be representative of the entire project. [6] MMF also allows
for the immediate delivery of basic working product to the end
user with the option to add features later … or not. Rockefeller’s
MMF was kerosene; he could have controlled the oil wells, pipe-
lines and rails (which he would do much later), but for at least
the first few decades he concentrated on what he could deliver
quickly and efficiently. Hamilton’s MMF was the preservation
of the Union under the Constitution, and he quickly curbed his abo-
lationist tendencies so that the South would support both.

Agile projects are subject to additional constraints not nor-
mally associated with development projects. These constraints
mainly involve tasks that fall outside the scope of the project
work, including:

—How to effectively manage an organization’s internal per-
sonnel so that the appropriate stakeholders are available
throughout the course of a project.

—How to gather and prioritize the most important features
desired by the organization throughout the ongoing
development cycle.

—How to adjust the notion of needed training in a continual
release environment.

—How to ensure customer team members are informed by
the full breadth of stakeholders required for enterprise
acceptance.

—How to secure timely approval of new technologies that a
team would like to leverage.

—How to address stakeholder discomfort with cultural, busi-
ness, social or other nontechnical changes related to Agile
software implementation.

Project teams are encouraged to consider these issues, as
well as mitigation strategies employed to resolve the associ-
ated risks, when considering Agile requirement elicitation
policies and procedures.

Honesty, Integrity and Agile Servant Leadership

“If your actions inspire others to dream more, learn
more, do more and become more, you are a leader.”

—John Quincy Adams

and Douglas Smith define a real team as “a small number of
people with complementary skills who are committed to a com-
mon purpose, performance goals and approach for which they hold themselves mutually accountable." This and other research on the subject defines a high-performance team as one that exhibits all these traits plus self-organization and direction, traits typically exhibited by mature Agile teams.

Members of such empowered teams are freed from command-and-control management and can use their own knowledge to determine how to best do their job. Hamilton knew we needed a central bank and entrusted its foundation to people who understood banking on the European model. Rockefeller understood the need to turn murky oil dredged from the Pennsylvania fields into kerosene but left the creation and operation of the refineries to those teams that understood it. Empowering teams also enables organizations to tap into people's natural abilities to manage complexity, thus reducing or eliminating the need for teaching and coaching individuals. Instead of providing detailed task lists, leaders of Agile teams should describe the iteration goals at a high level and let the team determine how best to accomplish the work. This self-organization naturally leads to self-directing teams that work together to create team norms and to make their own local decisions (i.e., at daily stand-ups).

This recognition that the team is in the best position to organize project work is liberating and motivating for team members. This practice alleviates many of the technical blockages seen in push systems where the task list and sequence is imposed on the team. Agile promotes a servant leadership model that recognizes that it is the team members, not the project manager, leader, coach or Scrum master who get the technical work done and achieve business value.

Many Agile project managers struggle to comprehend their role as servant leader. I’ve spoken with a number of PMs who basically told me that it wasn’t their job to “carry water” for the team. My advice to these prima donnas was to suck it up, forget about hierarchy and become a member (rather than the dictator) of their team. The servant leadership management approach redefines the leader’s role in relation to the team. It focuses the leader on providing what team members need, removing impediments to progress, and performing supporting tasks to maximize the team’s productivity. This makes sense in a technical environment where the product owner grooms and prioritizes the backlog and the team meets daily to resolve risks and issues—all tasks that would normally fall under the domain of the project manager in a traditional environment.

There are four primary duties a leader performs in this role as servant to the team:
1. Shield the team from interruptions.
2. Remove impediments to progress.
3. Communicate the product vision.
4. Provide the essential resources (tools, compensation, encouragement) team members need to keep them content and productive.

In addition to these four core duties, there are other activities that servant leaders should keep in mind: [8]
1. Learn the team members’ needs.
2. Learn the project’s requirements.
3. Act for the simultaneous welfare of the team and the project.
5. Have a vision of the completed product (understand the definition of “done”).
6. Use the project vision to drive your own behavior.
7. Serve as the central figure in successful project team development.
8. Recognize team conflict as a positive step.
9. Manage with an eye toward ethics.
10. Remember that ethics is not an afterthought but is an integral part of thinking.
11. Take time to reflect on the project.
12. Develop the trick of “thinking backward.”

“Thinking backward” means visualizing the end goal and then working backward to determine what needs to happen to get there and what problems and risks may arise. It may seem strange to limit a discussion of requirements development and management to decomposition, prioritization and servant leadership. Remember, however, that most development problems begin with requirements, and how they are broken down into their component parts, prioritized for value, and managed can mean the difference between high quality, increased velocity and stakeholder dissatisfaction.

The Agile team must be trained at project inception and coached by a knowledgeable advocate who can help guide them through the process. It is important that the Agile PM sets aside funding during the early acquisition stages for initial and ongoing training and support. Team formation typically follows the Adaptive Leadership [9] stages of forming, storming, norming and performing associated with the project management activities directing, coaching, supporting and delegating. Not all teams go through these phases in a predictable way, because each team is different. The best an Agile PM can do is to be aware of these models, look for signs that the team is in a particular phase, and then lead accordingly. While such general pointers are useful, the Agile PM should never expect teams to proceed in an orderly fashion and follow the traditional stages.

Rockefeller well understood the concepts of high-performance teams and servant leadership. It’s very hard to establish the world’s largest trust without such characteristics. He got in trouble for it with two presidents, but more on that later. Hamilton, though, often adopted a “my way or the highway” approach, drowning his critics in words and paper. It was only through the intervention of Washington that Hamilton lasted more than a day in office.

**Configuration Management and Change Control**

“The guy who knows about computers is the last person you want to have creating documentation for people who don’t understand computers.”

—Adam Osborne

The typical Agile practitioner will refer you to the Agile Manifesto (it’s their Bible) whenever possible, especially regarding this aspect of project management. They’ll quote “working software over comprehensive documentation.” They really, really hate documen-
tation, arguing that code comments are sufficient or that, due to the simplicity and inherent structure of modern code, there is no need for separate documentation. As an Agile PM, some of your greatest feats will be convincing Agile practitioners to document anything outside of code at all and convincing them that 'minimal documentation' is not 'maximum invisible documentation.' In a traditional development methodology, change control meetings are used to review and manage change. This is needed because the project is long enough that it has passed a requirements baseline phase and changes must now be managed. Because Agile works in short iterations, change control is not needed and becomes embedded in the iteration planning session. Of course, changes can be identified at any time and should be captured and documented. However, the changes should be discussed during the next iteration planning session and not during an iteration of work. There are times that a change stops the iteration due to its nature, but this is rare.

**Team Composition and Management**

*“Individual commitment to a group effort: that is what makes a team work, a company work, a society work, a civilization work.”*  
—Vince Lombardi

Team composition on Agile projects needs to reflect throughput maximization. You staff the team to be able to deliver the most potentially deployable quality software at the conclusion of every iteration/sprint with as little waste as possible. The ideal combination of team composition and management involves such factors as advocacy, communications, consistency and continuity. Small teams are preferred; in fact, at the beginning of John Adams' first term in office, people were astounded that the federal government had ballooned to 35 personnel, including Adams and five cabinet members.

A 1995 study [10] describing how merchant marines bring their ships into port used the term “distributed cognition” to describe how the crew operates as though using a common brain with distributed components. This phrase helps us understand why proximity and collaboration are so important on Agile software projects. [11] Each person on the team is busy forming a slightly different idea of what problem he or she is solving and how the solution should look. Each is encountering problems that someone else on the team might be able to help with.

Viewed in this way, we see software development as a problem of mental search and synchronization. We add the difficulty of resolving differing opinions and learning how other people work, what motivates them, what angers them, and so on. We see, from this perspective, why communication and collaboration are so important on software projects, and especially within short Agile timeboxes.

The Allen curve [10] is a graphical representation that reveals the exponential drop in frequency of communication between engineers as the distance between them increases. During the late 1970s, Thomas J. Allen undertook a project to determine how the distance between engineers’ offices affects the frequency of technical communication between them. That research produced what is now known as the Allen Curve. The research indicated a strong negative correlation between physical distance and the frequency of communication between work stations. The study also reported that communication drops off at about 10 meters in distance — basically, people won’t walk farther than the length of a school bus to ask a question. For this reason, close proximity and co-location are essential for Agile team success whenever possible.

Another challenge is keeping high-performing Agile teams together long enough for them to achieve peak performance. For this reason, Agile teams do not flourish in a matrix environment, where developers and testers are pooled and report to a separate entity. However, while Agile teams do well in projectized environments where the entire team reports to a common entity (i.e., the project manager), the developers can still change at the end of a contractual period of performance. The continuity of an Agile team enhances the “tacit knowledge” of the program — knowledge that is not written down but is instead supported through collective group knowledge, which tends to improve overall performance. It is therefore essential that every effort be made to either keep a high-performance team working together or, at a minimum, ensure that a core constituency remains even when other personnel are on the move.

Hamilton and Rockefeller both understood the need for co-location. Rockefeller based his trust in Cleveland (near the oil fields) and later New York (near the money). Hamilton’s greatest concern and victory was for a centralized government rather than a loose confederation of states as proposed by Jefferson and Monroe. He understood early on that the country couldn’t long exist without central executive, legislative and judiciary branches of government to handle money, taxes, war and the general good on a grand scale. No large country in history had succeeded in democracy without sliding into dictatorship, as France would prove in 1804. He then managed the deciding vote to move the capital from Wall Street to Washington, D.C., by promising the Pennsylvania congressional representatives a temporary 10-year capital in Philadelphia (who assumed they’d never build a capital out of that swampland next to the Potomac), thus exhibiting the traits of collaboration over contract negotiation. He also considered individuals and interactions by placing the planned capital next to the Potomac near Mount Vernon, thus gaining the support of George Washington.

**Scope and Estimation: Planning Poker**

*“Informed decision making comes from a long tradition of guessing and then blaming others for inadequate results.”*  
—Scott Adams

Probably the greatest concern for both Rockefeller and Hamilton was how to manage that huge scope of money and power, respectively. Rockefeller did it through double-entry accounting — it was said that even in his seventies he could quickly scan a lengthy ledger and spot a one-penny mistake. Hamilton managed the scope of his endeavors through lengthy reports and three-hour speeches. (OK, he wasn’t completely Agile, and he was a lawyer.) Similarly, one of the biggest questions a PM
will get from Agile practitioners is why they should estimate the current iteration at all. They see it as a waste of time since (a) they will eventually break down all their tasks into one-hour increments, and (b) anything they don’t finish in this iteration will be finished in the next. This is fine if you’re a code dork who communicates with little more than a screen and is responsible only for churning out code; it’s not so fine for a PM who has to communicate with the sponsor and explain what is complete, what is yet to be complete, and what will likely not be addressed.

There are many estimation techniques (i.e., Wideband Delphi, White Elephant, Affinity) that work just fine for short Agile iterations. But given the aversion of most teams to estimation at all, Planning Poker affords the most involvement and, therefore, accountability. Planning Poker is a consensus-based game where team members individually estimate user stories using numbered cards. This is followed by iterative discussions until consensus is reached. A variation of the Wideband Delphi technique, team members play their estimate cards face down prior to any discussion. Planning poker minimizes anchoring—the tendency of strong-willed individual voters who may have specific agendas that exert undue influence on the team (we’ll call it the Jefferson effect).

Story estimates may be done in actual days, ideal days or story points. All team members have identical card decks containing the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21, 34, 55, etc.). Story points are preferred and are used to estimate larger pieces of work (i.e., user stories and epics). If your cards include Fibonacci numbers over 21, it’s time to break down your tasks to more manageable chunks.

The process of the Planning Poker session is as follows:

- The meeting is facilitated by a Scrum master from another team who does not play.
- For each story, the team asks the Scrum master questions and discusses assumptions, constraints and risks.
- To vote, each team member plays a card face down that represents his story estimate. This is followed by discussion, during which no mention of the unrevealed story estimate is made. Then all players simultaneously reveal their cards by turning them over.
- Outliers (i.e., the players with highest and lowest estimates) are given an opportunity to explain their estimates to the team.
- The process continues in an iterative fashion until consensus is reached for each story.
- If stories are deemed too large to be included in the iteration, they may be divided or deferred to a later iteration.

**Definition of Done**

I was a student of Frederick Taylor’s “Time and Motion Studies” in college. One day, the professor was teaching us how to time a worker’s motions using a stopwatch: reach, grab, lift, return, drop. He told us, “Gentlemen, make no mistake of it. While you’re timing that worker’s motions, he hates your guts.” I mention this because Agile practitioners generally point to the Agile Manifesto or Principles (do they memorize these or carry them around in their pockets?) to argue against the need for metrics and monitoring. They’ll argue for “working software as the primary measure of progress,” the need for simplicity, welcoming change and maintaining a constant pace (for which metrics would be a roadblock). This is fine for a product owner who understands how to read a burn down chart and maintains constant vigilance on the product backlog. Unfortunately, that PO has a boss who has a boss, neither of whom understand anything but numbers. This is why senior management loves EVMS (i.e., SPI/CPI = 1.0 good, anything else is probably bad).

While seemingly simple, explaining a burn down chart to a senior manager can be like trying to explain alternate side of the street parking to a cranberry.

Agile methods place a greater emphasis on products delivered than on attributes of the process used to create those products. Measuring durations, costs and schedule performance — while not unimportant — tends to receive less attention in Agile approaches. Counts of delivered story points (embodied in working software) are the most prominent building blocks in Agile metrics as a byproduct of coding. One of the greatest benefits of Agile, due to its iterative nature, is its ability to allow for changes by the sponsor not only between but also during the short iterative cycles. Since Agile produces working software every 30 to 90 days on average (and in many cases, every one to two weeks), it could easily be argued that all Agile projects operate in the maintenance and sustainment phase of the software life cycle, where user demands trump any requirements agreed upon in the initial acquisition. This is especially important where safety of life (or, for financial clients, safety of money) is involved, where the inability of projects to turn on a dime can have catastrophic results in fielded software.

Consequently, of particular concern is the reporting and milestone requirements typically levied against Agile contracts, which may be especially difficult to achieve using Agile methods. As an example, consider an Agile team entering a preliminary design review (PDR) where the main concern is to ensure that the sponsor review team understands what is and is not included in the PDR. Requirements assigned to future iterations would have no detailed designs available, whereas in a traditional PDR all designs are typically completed. The selection and implementation of appropriate contracting vehicles to support the types of practices that successful Agile projects exhibit is a particular management concern. It is therefore important to concentrate on those Agile challenges and novel approaches when the traditional milestone review collides with Agile development.

**Monitoring and Control**

As an example, consider an Agile team entering a preliminary design review (PDR) where the main concern is to ensure that the sponsor review team understands what is and is not included in the PDR. Requirements assigned to future iterations would have no detailed designs available, whereas in a traditional PDR all designs are typically completed. The selection and implementation of appropriate contracting vehicles to support the types of practices that successful Agile projects exhibit is a particular management concern. It is therefore important to concentrate on those Agile challenges and novel approaches when the traditional milestone review collides with Agile development.

This is where that elusive concept “definition of done” needs to be established to the satisfaction of both the team and the sponsor in support of progress reporting. An Agile story is done when the function of that story is demonstrated to the customer and the customer agrees that the functionality substantially fulfills what is expected. [12] Given the incremental nature of Agile work, the most productive approach to technical reviews will also be incremental in nature, although an incremental approach to reviews may not always be possible.

One popular approach to incremental reviews in Agile development is test-driven development (TDD). [13] In this approach, rather than developing tests to verify an already built code base,
test cases are developed first (and often automated) and the code is written to pass these tests. In this scenario, technical progress can be understood through metrics that show the pass/fail outcomes of the tests. In fact, for most Agile projects the ‘definition of done’ is the successful passing of established acceptance tests. For many Agile projects the simplest and most accurate representation of progress is “story points promised vs. story points delivered” (which roughly translates on a traditional project to “estimates vs. actuals”), since delivered features are a composite of story points.

Earned value is one of the primary tools that the Department of Defense uses to measure contractor performance. For programs valued at more than $20 million, an earned value management system (EVMS) is required to be used, and for programs more than $50 million, a validated EVMS must be used. “EVM techniques, however, assume complete planning of a project to discrete work package levels, then assigning cost and duration to these packages.” [14]

AgileEVM calculations are based upon delivery of completed, tested units of functionality. No credit is given for delivery of intermediate work products. Therefore, while AgileEVM may be seen as incorporating quality standards into the metric and may be seen as providing stricter evidence with respect to delivery of value, proponents of traditional EVMS may need to be consulted on the difference between performance indexes that measure product progress (traditional) and value enhancement (AgileEVM).

Measurement and Analysis

Start by specifying measures. At their core, Agile methods are “inspect and adapt” methods. Inspection implies measurement. However, there is very little explicit Agile industry guidance on how and what to measure. While each Agile PM must determine metrics applicable to their practices, such metrics should be selected to provide transparency, improve effectiveness, and enable continual improvements. Most importantly, the PM should consider the metric contribution to the value stream. If it doesn’t directly or indirectly provide value to the customer, collection and analysis of the metric should be challenged. Potential measurements for an Agile IPT may include the following:

- **Burn down**: Keeps track of remaining effort; encourages early course correction.
- **Velocity**: Amount of product backlog a team can complete within a given iteration; provides a basis for planning and estimating (in some cases “story point” collected and completed can be used to determine iteration velocity). Remember that velocity between teams is not comparative or additive, since it does not show relative performance or productivity of teams.
- **Churn**: This metric shows how many lines of code the team added, deleted or modified during the most recent iteration. With the code churn metric, you can analyze how source files are changing over time.

Each Agile PM should spend time determining which metrics are most applicable to their unique practices. Metrics should be selected not only for transparency, effectiveness, and enabling continual improvements, but also with regard to the strategic goals of the organization and the project itself. In addition to analyses conducted at the review and retrospective meetings, Agile teams typically perform metric analysis at the daily team meetings where iteration and release burn down data are analyzed.

Velocity constitutes one of the more useful Agile analyses. Solid Agile teams have a consistent velocity (+/- 20 percent). Fluctuations in the velocity metric may indicate a need to stabilize a team or environment in terms of training or tool usage. It’s important to note that velocity is not to be used as comparison between teams, since it does not show relative performance or productivity of teams. Velocity is a measure of the number of features developed by an individual or team during an iteration. It does not account for size, and therefore comparing a velocity of 20 one-hour features with one of six eight-hour features can be misleading at best.

**Agile Product Integration**

“There are only three things that can happen when you pass the ball, and two of them are bad.”

—Paul “Bear” Bryant on risk management

The Agile approach toward integration should be to do it continually, and therefore documentation should be done continually throughout the life cycle. The general Agile rule is if every developer integrates and commits every day to the baseline, a big system could be developed by slicing the products in such a way that many integration points are planned. Therefore documentation should occur once a week or once a month, depending on the project. Of course this creates overhead, but risk is also sliced and minimized throughout the project. Remember that Agile favors working product over lengthy documentation, not instead of. This is a common myth that can lead to project failure when documentation is insufficient to properly describe integration planning.

Agile integration procedures and criteria should ideally be generated automatically if they are to be useful in a fast-paced development environment. One potential artifact that might satisfy these criteria is the “Agile functional specification,” especially in the form of user stories that evolve in conjunction with the client or their business representative. This implies that as the user stories evolve, the technical specifications will evolve as an automatic derivative. To achieve this, the user stories must be written in a context-aware structured syntax — in other words, a domain-specific language (DSL). Then it will be possible to consume the user stories and, from them, auto-generate the interfaces and the unit tests required to create an Agile technical specification. The interfaces are derived from the user story setups and the unit tests from the user stories.

If Agile is to succeed, testing (e.g., TDD) must become a central pillar of the development process. Incorporating testing at the end of a major project will only lead to show-stopping bugs and major timeline setbacks. To ensure that the product is high quality, Agile PMs should include test personnel from the very beginning so they can spot issues early on and prevent major problems down the line. This is a major component of Agile, and process infrastructure should be in place to record test results throughout the product life cycle.
Agile Release Planning, Prioritization and Tracking

The “iron triangle” of cost, time and scope is fundamental to traditional release planning. An Agile perspective on this triumvirate is expressed in the following equation:

\[ \text{Time} \times \text{Capacity} = \text{Scope} \]

where

- **Time** = Number of iterations \( \times \) iteration length
- **Capacity** = Average velocity per iteration
- **Scope** = Total number of story points that can be completed in the release

Using the above equation, a team with an iteration length of two weeks and an average velocity of 30 could complete 300 story points in approximately 10 iterations or 20 weeks. While seemingly straightforward, this equation must be understood within the context of the Agile approach to project scoping. [15]

It is important to note that even after stories have been broken down and estimated, they are not generally specified to the degree that would be found in a traditional requirements document. This does not necessarily imply increased risk, because successful Agile teams rely on ongoing dialogue with users, user proxies, and subject matter experts throughout the course of the release in order to gain insights needed to satisfy the users—usually better insights than could be gained from typical requirements specification documents. If the user interaction that makes this dialogue possible is missing, the prioritization of stories across iterations is another important aspect of release planning. The following four factors are critical considerations during prioritization: [16]

- **Value of the story.**
- **Cost of developing the story.**
- **Knowledge generation regarding requirements, the domain, user needs and the underlying product technology, architecture and design.**
- **Risk (including technology, business, schedule, cost and functionality risk).**

While a certain amount of prioritization will take place during initial release planning, on Agile development projects, prioritization is ongoing and stories are often reprioritized at the end of each iteration. Successful adoption and execution of this dynamic approach to prioritization once again requires a close relationship and ongoing dialogue with program stakeholders.

One of the most commonly used charts for tracking progress on Agile releases is the release burn down chart. On the release burn down chart, the x-axis expresses iterations, while the y-axis expresses story points remaining to be completed. The chart provides an early indicator of potential future issues, but only discussions with the development team will reveal the reason for discrepancies and what actions, if any, need to be taken.

As with any other progress tracking method, using user stories to generate velocity measures can lead to some anomalous results. For example, if the user story sizes vary widely between iterations, velocity could appear lower than is warranted. This sizing difference could also result in one story taking an inordinately long time to complete, possibly even resulting in a velocity of zero. A development team should develop its own norms in terms of the relationship between the number of stories, the number of team members, and iteration duration. In other words, teams need to be agile.

For Hamilton, apart from saving the Union, his highest priority was banking. The Bank of the United States enabled the government to make good on four powers cited explicitly in the Constitution: the right to collect taxes, the right to borrow money, the right to regulate trade among states, and the right to support fleets and armies. I’m guessing very low on his backlog was meeting Vice President Aaron Burr for a duel on a lonely sandbar across the Hudson from New York. And just as Agilists argue over the explicit meaning of “processes and tools” and “interactions,” Hamilton had to deal with the Constitutional meaning of such minutiae as “necessary and proper.” In a memo to Washington in 1791 regarding the banking issue, he indicated that “necessary” didn’t mean “indispensable” so much as “appropriate.” Rockefeller’s backlog most assuredly was headed by “acquiring wealth;” since he equated wealth with his own Baptist concept of charity (that part he didn’t keep for himself).

**So Do You Still Want to be an Agile Project Manager?**

“The two most important days in your life are the day you are born and the day you find out why.”

—Mark Twain

At this point I feel it necessary to resurrect Hamilton and John D., having been somewhat critical. I’ll pass on Hamilton, since he’s riding a popularity crest right now with a Broadway play, avoiding being erased from the $10 bill — and hey, who doesn’t love a treasury secretary who can rap?

John D., however, is known as a robber baron and thief who enriched himself at the sake of the poor. In reality, he sold kerosene at a reasonable price to the poor, who previously sat in the dark until the sun came up. And most of what they called “crimes” in those days would be called “loopholes” today. An abolitionist who supported the Underground Railroad, he endowed the University of Chicago as the first Midwest Baptist college for their...
first 10 years. At the request of nine African-American women conducting school in an Atlanta basement, Rockefeller built and continued to support in perpetuity Spelman College. He named it after his wife's family — he was never known for putting his name on things (unlike Carnegie). As a result of losing the 1911 antitrust suit, Rockefeller found himself owning about a quarter of 34 companies instead of one. His net worth increased in that year from about $300 million to $1 billion. Exhibiting the Agile benefits of distributed cognition, those 34 companies skyrocketed in value (e.g., Standard Oil of New Jersey, New York, Indiana and California became Exxon, Mobil, Amoco and Chevron, respectively). By 1922 Rockefeller had given away over $475 million to charity — more than double what Carnegie had donated — proving that you can be Agile, wealthy and a philanthropist.

People have wanted to control things and other people since the dawn of time. If that is your goal, steer clear of Agile projects, where your role will be reduced to that of a servant leader by what is essentially an autonomous collective of self-directing and self-organizing individuals. There is currently no comprehensive project management model to understand and analyze the entire spectrum of Agile innovation, interactions and processes. After accepting the reality that one size does not fit all, practicing PMs may need to rely on a combination of models and processes to understand the extent of innovation in Agile projects and find the optimal way of managing them. If, however, you can check your ego and accept your role as a team member with honesty and integrity, you may be as successful as Alexander and John D.

REFERENCES


ABOUT THE AUTHOR

Peter D. Morris, PMP, PMI-ACP is a Process Engineering Consultant for INDUS Technology Inc. under contract to SPAWAR Systems Center (SSC) Pacific. He has worked in both the commercial and DoD/Federal sectors, including several Fortune 500 companies. He has authored numerous technical reports, including publications for the NSA, DNA, US Army NTC, USAF Space Command and the SINCGARS & JTIDS communications programs. Having managed or consulted on over 50 Agile projects in both the government and private sector, Mr. Morris has also been instrumental in establishing successful PMOs at numerous organizations, targeting strategic Business Process Improvement associated with Project and Process Management, Engineering and Project Support enhancements.