The Fact that your Project is Agile is Not (Necessarily) A Cost Driver

Introduction

Use of agile practices is increasing in government projects. Agile projects rely on highly skilled development teams communicating with clients to deliver the most value for the money spent. This requires a mind shift on the part of both the developers and consumers of a software application to accept the reality that things will change over the course of the project.

Agile is a philosophy, not a specific development process. All true agile teams apply this philosophy to their projects and aspire to stay true to the agile manifesto. There are, however, many different frameworks for agile implementation which apply many different processes and practices. The fact that a project is agile may suggest a different approach to cost estimation over a more traditional development approach. Knowing a project is agile is not enough to inform an estimate without conversations between the agile team(s) and the estimating team of a project to determine what agile framework is being adopted and what specific agile processes and practices are involved.

In a truly agile environment, one might argue that an estimate is not necessary. Since the team size, iteration length and schedule are known quantities, the cost should be obvious. In reality, this is rarely the case. While many organizations are adopting many agile practices and consider their development environment to be agile, they are still working on contract to deliver specific requirements to a customer, often on a firm fixed price contract. Failure to estimate against these requirements could be disastrous. Furthermore, even when this is not the case, most organizations are not willing to launch a significant project without an understanding of the likely cost and schedule.

This paper discusses the agile philosophy and covers various approaches that organizations take to adopt that philosophy. This is followed by a study on why the claim that the mere fact that a project is agile is not enough to drive changes in a cost estimation (as compared to a more traditional approach for the same project). A methodology and rules of thumb incorporating agile practices that could indeed influence cost drivers is discussed. Common agile metrics are presented that will aid agile teams, customers, decision makers and program controllers to understand the progress of an agile project.

Agile Overview

The notion that it might be smart to build software in iterative or incremental chunks did not emerge with the *Agile Manifesto.* Iterative and Incremental Design and Development (IIDD) has been around for close to 80 years. The practice of time boxed product development cycles has been practiced by smart engineers from the Department of Defense (DoD), NASA, the US Air Force and in many pockets of industry.

The software industry, being much less mature than the hardware industry, suffered significant starts and stops through its early years and into the present. At its inception, software development was not treated like an engineering discipline but rather a combination of art and science with a bit of black magic blended in. As technology and tools improved, software developers and engineers have been able to tackle more and more complex problems with software. The problem was that few stopped to take a breath and think about software as a discipline. Increasingly complex software projects led to increasing instances of software project failures, often leading to expensive and embarrassing situations for both the developers and their customers. While many attempts have been made to harness software development in the face of increased complexity, agile practices have arisen as a viable way to attack the problem

In February 2001, a group of software development professionals got together and created the agile manifesto [1].

We are discovering better ways of developing software by doing it and helping others do it:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

That is that while there is value in the items on the right, we value the items on the left more

This is the primary philosophy that guides all agile projects. There is also a set of common principles that are intended to support teams in implementing and executing with agility [2]:

- *Our highest priority is to satisfy through early and continuous delivery of valuable software*
- *Welcome changing requirements, even late in development. Agile processes harness change for customer's competitive advantage*
- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale*
- *Businesspeople and developers must work together daily throughout the project*
- *Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*
- *Working software is the primary measure of progress*
- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely*
- *Continuous attention to technical excellence and good design enhances agility*
- *Simplicity – the art of maximizing the amount of work not done – is essential*
- *The best architectures, requirements and designs emerge from self-organizing teams*
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts behavior accordingly.*

To put this in simpler terms. At the beginning of a software project the leadership (Product Owner, Scrum Master, Business Owner, System Engineers, etc) review customer requirements ad translate them into user stories which become the backlog for the project. Customers work with leadership to prioritize the backlog and the agile team starts working on the highest priority items. Usable chunks of software are developed in short periods of time (sprints, iterations, etc.) and are delivered regularly for customers to review and critique. Following these reviews, the customer works with the team to re-prioritize the backlog as their requirements and/or perspective changes based on the current software or changes based on market or business priorities.

While the above description should provide the reader with the general operation of an agile project, it gives very little insight into how the team goes about achieving this continuous rhythm to deliver valuable software regularly. In fact, the intent of the manifesto is not to tell projects how to do this but rather to give projects a set of principles to follow to lead them to this eventual end. There are however a set of agile practices that are intended to offer a path forward. Some of the more popular agile practices include:

- Regular delivery of usable chunks of software that deliver value to the customer
- Pair programming – no production code is written without two sets of eyes and minds
- Continuous integration with automated tests – builds of the software are done frequently (one or more times a day) and a set of regression tests are run to make sure the quality is maintained
- Test driven development – developers write the test first, based on the story and/or task they are working on, then write just enough code to make that test pass
- Daily stand up meetings – the team gathers once a day, face to face, to discuss progress, plans and obstacles
- Co-located teams – teams work in the same location, generally open areas, to facilitate easy access to other team members
- Customers (or customer proxies) are co-located with and/or participate with the team daily
- Simple design – no solution should be more complicated than necessary to solve the current issue
- Refactoring – when a developer touches a piece of existing code they should strive to leave it in better shape than they found it

All truly agile projects embrace the manifesto and attempt to apply the principles as much as possible to their daily work. Many agile projects adopt one or more of the practices listed above, or other agile practices as necessary and prudent to the projects needs. Not all agile approaches are the same, there are various frameworks for agile implementations. Some of the more popular ones are listed here:

- Extreme Programming(XP) – XP values extreme communication between developers, customers and management. Pair programming and 100%-unit testing are key practices. XP incorporates five key values
  - Communication
  - Feedback
  - Simplicity
  - Courage
  - Respect
- Feature Driven Development (FDD) – development is driven from functionality perspective following the following steps
  - Develop the overall model
  - Create feature list
  - Plan by feature
  - Design by feature
  - Build by feature
- Kanban – eliminate work in progress to alleviate bottlenecks throughout the development. Kanban's most well known feature is a task board divided into three columns - To-Do, In Progress, Done. The following practices are applied:

    - Visualize the workflow
    - Limit work in progress
    - Manage flow
    - Feedback loops
    - Collaborate to improve
- Scrum – One of the most popular agile frameworks, scrum defines the team as product owner, development team, and scrum master. The most common scrum practices include:
    - Time boxed iterations
    - Daily stand up meetings
    - Regular deliveries of value to the customer
    - Retrospective meetings at the end of each iteration

Why the Fact of Agile is not enough to Predict Costs

Clearly agile is a paradigm rather than a practice, thus one would not expect that having an 'agile check box' in their estimation model would have the effect of increasing or decreasing cost and effort estimates for a development project by some specific percentage. However, this author recognizes that boldly stating something does not necessarily make it so. Additionally, the question lingers as to whether there may be some sort of causal relationship between a project being agile and the effort required to develop that project.

In an effort to draw data driven conclusions, a study has been conducted using the International Software Benchmark Standards Group (ISBSG) data repository for Development and Enhancement from 2019. ISBSG is a not-for-profit organization whose aim is to promote the use of IT industry data to improve software processes and products [3]. The current version of the ISBSG database has over 9,100 software projects submitted by leading IT and metrics companies from around the world. This study focused specifically on extracting a subset of this data and using a causal analysis tool to identify any potential causality between a projects 'agile-ness' and the development effort. The RapidMiner tool [4] was used to cleanse and normalize the data and the Tetrad tool available from the Carnegie Mellon Institute [5] performed the causal analysis. These tools are Open Source and are freely available to the community.

RapidMiner was used to massage the data set as follows

- Only projects with a Quality Rating of A or B were considered
- Project which were started before 2009 were filtered out (less than one percent of projects that reported agile methods occurred before 2009)
- Projects that had either Labor Hours or Functional Size = 0 were eliminated
- For the Agile Methods attribute– text answers were converted to numeric values (Yes = 1, No or no answer = 0)

Figure 1 Represents the process that was developed. The end result of this process was a csv file that could be fed directly into the Tetrad tool. RapidMiner also produced a correlation matrix that demonstrated very low correlation between all the attributes except Functional Size and Normalized effort.

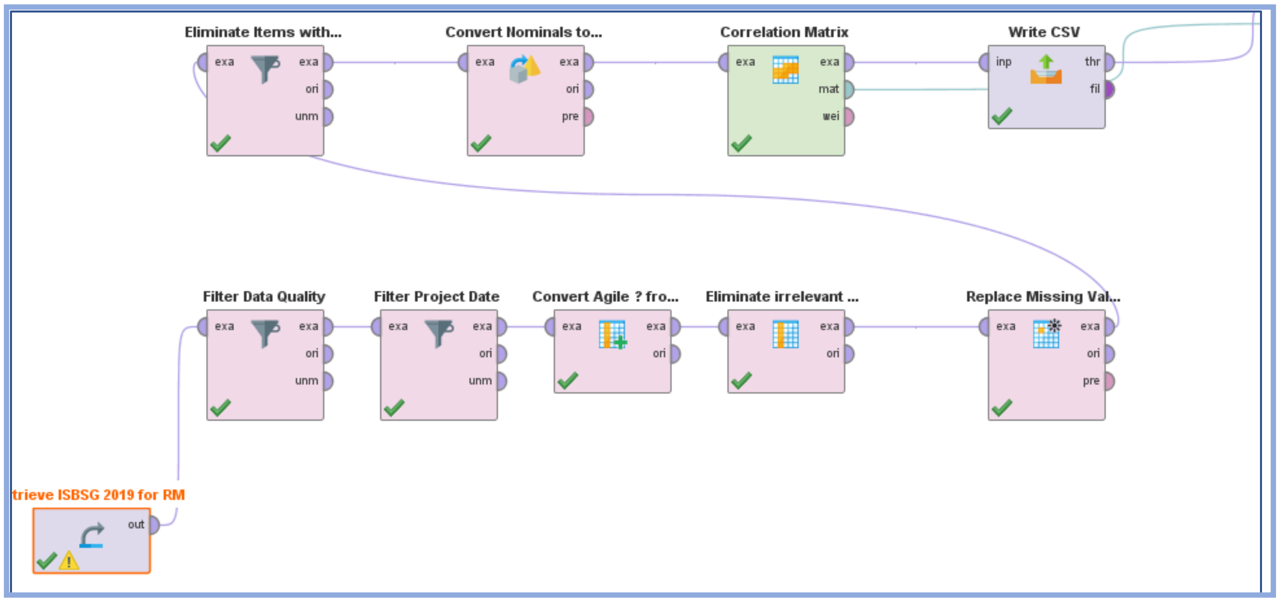

Figure 1 Rapid Miner Process for processing ISBSG data into Tetrad form

This data was then processed through the Tetrad System using the standard PC causal search algorithm which is constraint-based and assumes no confounders and its discovered causal information is asymptotically correct. This analysis was performed on the whole data set that met the above criteria as well as individually on each industry sector which contained agile projects. The creation of this process made it possible to study the ISBSG data for this study and many others with minimal additional time to change the aspects of the studies.

The results clearly indicate no causal relationship between "agile-ness" and effort. Figure 2 shows results from the full dataset. Table 1 contains a breakout of agile vs non-agile by industry sector.

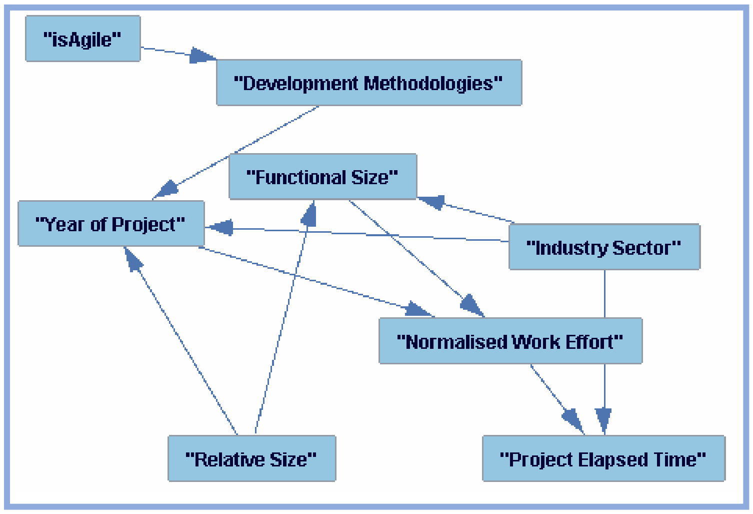

Figure 2 - Tetrad Causal analysis Results

Industry Sector	Non Agile	Agile
All	3181	345
Banking	85	29
Education	31	13
Government	223	86
Insurance	226	181
Professional Services	33	13
Utilities	14	5

Table 1 - Agile vs non-agile projects by Industry Sector

Clearly this shows that just the fact of agile is not enough for prediction. If should also be obvious that the use of some agile practices could and should have an impact on effort and cost for a given project. A great improvement to the ISBSG data collection process would be to extend the data collection targets for agile projects to include information about the agile practices employed and collect agile specific metrics for these projects.

Methodology and Rules of Thumb for Estimating Agile Projects

An agile software development project is just like any software development project in many ways- requirements, design, code and test all need to be accomplished – they are just approached in a different fashion than more traditional projects. From an estimator's perspective, especially in the early phases of the estimate – they are still required to understand the capability desired and from that the likely cost and schedule for the project.

There is a conundrum around estimating pretty much every software project – at the point at which the first estimate is required – there isn't nearly enough information about the project to make it easy. Every estimator knows that a good understanding of the project, technical and team parameters (when these are available) are instrumental in developing a credible estimate.

In this author's experience, one of the most frustrating things that estimators face is the difficulty in understanding what it means when the development team(s) declare they are agile. There is a wide spectrum of ways to interpret that. A truly agile team knows team size, length of iterations and schedule – they don't need an estimate as they will deliver what they have at the end of the schedule – which will be fine with the customer because they have worked closely with the development team to prioritize the most desirable set of features within the time constraints. But experience tells us, especially when government contracts are in play, that most agile teams don't meet the truly agile standard. This is in no way intended to be derogatory; the trend appears to be that many government contracting organizations recognize the value of implementing agile practices even though acquisition policies prevent them from being truly agile. This is because agile practices have been proven to add value to the process by creating a culture where the delivery of incremental value is favored over big releases, where customer interactions improve quality and customer satisfaction and where self-empowered teams are more likely to deliver quality solutions.

There are various ways to adapt an estimation methodology to the various degrees of agile-ness within this spectrum. This is particularly true in the early days of a contract/engagement when cost, effort and schedule are needed for planning or to support negotiations and to give decision makers what's needed to make wise decisions.

If we look at the far left of the spectrum, in some ways the estimate is in the hands of the development team. They know the team size, they know the length of iterations or sprints and they know the delivery date. Here the estimators job is to take their estimating methodology, apply it to what they know about the capability and schedule that the customer is expecting – taking into account how specific agile process may impact that methodology and offer the team advice on any potential pitfalls or risks associated the capability the team expects to deliver or the schedule within which they think they can deliver it. The estimating team becomes responsible for assessing the risk of delivering the minimal viable product the customer expects in the time frame the customer expects it. On the other end of the spectrum, where the customer still has a fairly unretractable set of requirements, with a firm fixed price contract and schedule, the estimators job isn't all that different from a more traditional project. Informed by the specifics of the agile implementation plans and practices, the estimator should use traditional methods to prepare an estimate. The difference between a more traditional scenario will be the fact that the agile team will have a good set of metrics regarding previous performance. From these metrics they should have their own ideas of practical schedule and cost targets. Serious deviations between the estimating team and the development team in their estimates should stimulate discussions that should be valuable in converging on a plan that aligns with the historical memories of both teams. Knowing that the project is agile, or is at least applying agile practices, gives the estimator an advantage if they know the right questions to ask to determine what the team means when they call their project agile. Some of the questions that may be pertinent are covered in the paragraphs below.

Agile teams tend to be highly skilled. It is not easy to be a slacker in an agile environment, as co-location and daily stand up meetings highlight situations where work is not being accomplished efficiently and successfully. In an agile team, the cream quickly rises to the top and shirkers are eliminated. As new members are added to the team, they are poised to take advantage of the rest of the team to quickly come up to speed on the projects. The estimator should consider higher than average values for model inputs related to team experience.

Another common attribute of an agile team is the fact that they tend to have quite sophisticated tool sets. These tools are needed to manage the agile process – keeping track of iterations, releases, epics and their corresponding metrics, as well as managing the backlog. They are also necessary for communicating the teams progress to the team, the customer and management. Tools can also help facilitate compliance to standards when such is required. The estimator should consider higher than average values for model inputs related to tool sophistication or automation.

The co-location of teams generally increases the productivity of the team. Not only are new members brought up to speed quickly, as mentioned earlier, there is also an increase in the urgency to meet shared goals resulting in greater team cohesion. When the team is co-located, questions can be answered in real time and no one languishes in their office or cubical trying to figure out a problem that another team member is quite likely to have a solution to. Additionally, productivity improves when customers, stakeholders and/or SMEs are co-located with the agile team. The estimator should consider higher than average values to inputs related to distributed vs co-located work environments and good communication practices.

Continuous integration with automated testing has great promise not only to increase the agile teams' productivity but also to improve the quality of the end product. Every team member should test their changes before they check in code but what they may not find as a result of their unit tests are unintended consequences realized when interfacing with other parts of the application. This practice involves frequently rebuilding the application – either once a day or once an hour or even upon the check in of changed or new code. Once the build is complete, a suite of regression tests is run and any errors are flagged. The team is expected to drop everything to make the tests return to green. The benefit of this is that problems are identified while the team still remembers what changes they made and should easily be able to quickly track the problem area and address the problem. The estimate should consider higher than average value for inputs related to the complexity of internal and external integrations.

An agile software development project is like any other software project in that there is code that must be designed, written, tested and integrated. What makes agile different is the way those processes are approached and the transparency and flexibility of those processes. There are agile practices that if implemented correctly are likely to improve productivity. There are also agile practices that might reduce development productivity in the interest of improved quality of the finished process. Modeling methodologies should certainly be extended to incorporate new pieces of information available to the estimating team, but it is also important to respect the fact that many factors transcend the choice of an implementation approach.

Useful Agile Metrics

One of the huge benefits of agile, both for the development team and for those responsible for project control, is that the processes associated with most agile frameworks are constantly being measured. An additional bonus is that most agile toolsets put a large emphasis on measurement and tracking – making it easy for the team to record important measures and making it easy to turn those measures into valuable information the organization can use to manage projects.

As with all software projects, metrics such as effort and schedule progress (compared to plan) are important for project control. There are other metrics associated with agile projects than can also be of value both to project control people, but also to the estimating and development teams. Metrics at the sprint/iteration level are valuable to the team but maybe less valuable at the project control level as team metrics relate specifically to the team. In agile teams, measurement is often very team specific. It is however possible to aggregate these metrics through weighting to make them valuable at the feature or epic level.

Many agile teams have a retrospective at the end of each iteration or sprint. This is the opportunity to review what was good and what was bad about the previous two weeks (or whatever the length of the sprint was). Often this is also a time for them to review sprint related metrics such as:

- Velocity – the amount of effort estimated for the stories completed within an iteration. In general, agile teams only take velocity credit for stories that are 100% complete at the end of an iteration. Velocity is often a determining factor for the team on how much to take on for the next iteration.
- Burn down chart – this is used to track the progress of completed features, user stories or story points completed vs those planned.

- Burn up chart – this is used to track progress toward the final goal of completing all the features in the backlog. The burn up chart also shows progress against plans
- Defects injected/Defects removed/Latent defects – the number of defects that are injected during an iteration, the number that are removed and the number that are left to be detected postproduction.
- Cycle time – average time a team spends working on a user story
- Stories or story points per iteration.

Metrics collected at the regular retrospective are useful to keep the agile team(s) focused, to inform future iteration plans, and (hopefully) to help the team achieve a regular cadence for delivery of features. There is also a need for metrics that inform management and customers of progress to date against plans, Things like velocity and burn down/up can help inform these decisions though this can be complicated if there are multiple teams involved since each team could potentially have their own version of what constitutes a story or a story point. This can be accomplished if the program, in conjunction with the agile teams, agrees on an effective weighting system that makes it possible to count work planned and work accomplished in the same units. Once common units have been established, metrics that are useful at the project control level include:

- Epic/Release Burn up chart – this chart shows progress against plans on the feature level for releases and or epics.
- Epic/Release Burn down chart – similarly this chart shows progress against plans for completing the features and capabilities planned for a release or epic.
- Product backlog – this is a measure of the amount of stories or features currently still in the backlog.
- Changes in product backlog – this is a measure of how the product backlog changes from release to release – a good indication as to whether there is churn in requirements.
- Defects injected – number of defects that were injected during the development of a release
- Defects removed – number of defects that were identified and removed during a release
- Latent defects – number of defects that were detected post release.
- Cumulative flow diagram – this is an area chart that shows the project of work items for a specified time period. It is a tool to help visualize project progress and to identify potential issues or problems. Not unlike a Kanban chart, the cumulative flow diagram shows progress, backlog (to-do) and work items currently in progress.
- Number of features delivered – this is a measure of the number of completed features that have been delivered. This (tempered by a weighted size measurement system) will help with ascertaining work accomplished vs work planned.

Conclusions

The agile paradigm is here to stay. There are many reasons why applying agile practices to a project may help revolutionize software development. The current practice of agile, particularly in the government, appears to be limited to the adaptation of many agile practices without fully embracing all the tenets of the agile manifesto. This is an entirely reasonable approach that is likely to enable development teams working on government (and other) contract to realize many of the benefits of agile without risking the consequences of failed contracts or unmet expectations.

With this in mind, it is imperative that estimators perform estimates for projects even if they are intended to follow the agile paradigm. These projects are still software development projects. Contractors are being paid to deliver a set of requirements (minimum viable product at least) and they need to commit to how much time and effort it will take them to deliver that capability. Agile development teams may speak a different language than the traditional development teams so there may be some onus on the estimator to understand this new language, much as there should be an onus on the agile teams to help create the Rosetta Stone that will facilitate communication. Estimators should consider that, in principle, agile projects are like any other, but there are agile practices that are often employed that could have productivity (thus cost and effort) implications and should consider it fair game to include the use of these practices into their estimating ground rules and assumptions.

References

[1] The Agile Manifesto available at https://agilemanifesto.org/ retrieved January 2020

[2] The 12 Principles Behind the Agile Manifesto available at https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/ retrieved January 2020

[3] www.isbsg.org

[4] www.rapidminer.com

[5] http://www.phil.cmu.edu/tetrad/about.html