ENVISAGE TECHNOLOGIES

ADOPTING AGILE

IN A GOVERNMENT CONTEXT

Michelle Cole, COO ENVISAGE Technologies Corp.

January 24, 2013

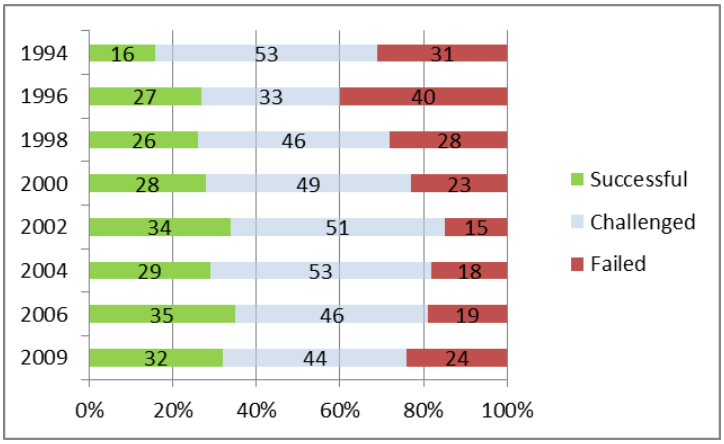
Background

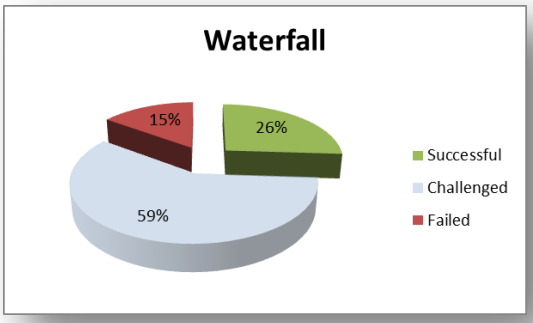
Envisage Technologies converted to using Agile methods in 2006. We have worked with a variety of Federal, state and local governments to implement software solutions, always using agile methods when software development was needed.

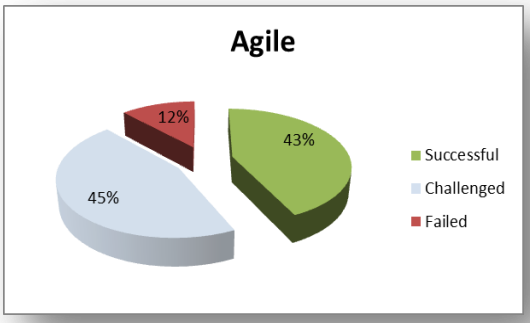
One of our contracting officers posed the following questions:

1. What would you recommend to agencies in terms of cultural change that would assist them in implementing agile development?
2. What contract type would you think best lends itself to the agile environment?
3. What would you recommend for evaluation factors?

Perhaps our contracting officer has seen the CHAOS reports by The Standish Group International, Inc., wherein IT projects are analyzed. The rate of success (measured in time, budget and features) is gloomy.[1]







Standish Group's research shows that Agile projects have significantly greater success. This white paper is intended to help those that are interested in transitioning their projects to Agile.

[1] http://zenexmachina.wordpress.com/2012/07/19/waterfall-vs-agile-a-knowledge-problem-not-a-requirements-problem/; http://www.swqual.com/verification_validation.html

Culture change

Focus on the business value

Agile software development is focused on business value. The final deliverable is improved business outcomes, not just a software product. So, when asked about what is different about Agile development, I suggest looking at the Agile Manifesto (agilemanifesto.org).

The Agile Manifesto reflects a significant difference in cultural values:

> We are uncovering better ways of developing software by doing it and helping others do it.
> Through this work we have come to value:
>
> 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, while there is value in the items on the right, we value the items on the left more.

The authors of the manifesto witnessed items on the right (processes, tools, documentation, contract negotiation, plan following) becoming more important than delivering business value and wanted to change that balance. While the items on the right might be necessary to accomplish the goal, if the items on the left (valuing individuals, working software, collaboration and change response) were missing, the projects failed.

Identify the smallest list of most helpful features and welcoming discoveries

One of the most noticeable differences in an Agile project is delivering features in small increments. The goal is to produce the smallest feature that offers the most business value and get it in the hands of the users. As the feature is used, new priorities emerge. Agile welcomes this.

In traditional software development, the customer tries to ensure that the specifications are very clearly and specifically enumerated and the vendor wants the customer to sign off on all of the features as the full scope of the project. First, when users see the software, they often have expectations that are not in the specifications or the feature is not implemented as desired. Second, personnel change, so expectations change too. Third, environments changes. At that point, the contract becomes the focal point of the project, because the dollars have been mostly spent and the business value has not been realized.

Agile developer/customers recognize that the Pareto principle (more commonly known as the 80/20 rule) holds true in most software. Eighty percent of the features go mostly unused while twenty percent of the features get the majority of the usage. When people are designing theoretical software (creating specifications about what they want new software to do) many features are listed. As users begin to use the software, they often discover needs that are more critical than some of the features listed in the

specifications. If features can be developed incrementally and feedback can adjust priorities, considerable spoilage (features designed but not implemented) and unused features can be eliminated.

Agile developers/customers believe perfect specification documents cannot exist for large projects, because requirements will emerge as knowledge of the domain is uncovered and because no business is static in its needs. So, Agile breaks down features into small pieces and keeps a running list of their priority. The process is to work the list from top to bottom, keeping the Work In Progress (WIP) to the smallest amount possible. This allows for constant feedback and easy rearrangement of the list, resulting in the most beneficial spending of the customer's dollars. Very little work is ever thrown away and features that exist are the most important features. Funds are not used on the rarely used features, unless resources are available at the end.

This philosophy of incremental delivery is sometimes uncomfortable to people who have been involved in traditional software development.

Prepare to spend Subject Matter Expert (SME) time each week

Reasons Agile software is more successful are increased clarity of communication and increased accountability because of the frequent feedback loops. The customer being involved is one of the most critical loops.

At regular intervals (generally once a week, once every two weeks or once a month), an Agile software development team will meet with customers. In that meeting, they will demonstrate all features that were completed since the last meeting. The customer will provide feedback – what is exactly as expected and what will need to change as a result of a better, shared understanding. The customer will also choose the next priorities for the team. The customer will need excellent knowledge of the domain and an ability to make decisions. A good project manager on the Agile team will help the customer understand dependencies of some features on other features and architectural risks (that should be handled early to minimize rework).

For subject matter experts usually accustomed to the traditional software development method of documenting detailed specifications and not seeing the software until the end of the project, it is often a surprise to have such frequent interactions with the development team. The payoffs for this high level of involvement include

1) a clear understanding at all times about which features are complete,
2) a high level of quality in the application (because features are going through quality testing at regular intervals to demonstrate them to customers during frequent team meetings), and
3) features meeting the business need, deployed as soon as possible.

Rethinking what is valuable

How do we know when software projects are on track or succeeding? For many traditional software projects, this evaluation is based on the volume of paper (project plans, specifications, test plans, traceability matrices) that is produced right up until the final deliverable.

Agile questions this evaluation. Are these documents adding business value or are they just a byproduct of the process? In one year, will the requirements documents still be valuable? What project plans will still be valuable? How often will the test plans and traceability matrices get out of date?

Instead, Agile asks what can be incorporated into the software so it is continually adding to the business value. Regular demonstrations, regular releases and shrinking priority lists minimize the work needed for most project plans. Specifications are built into automated test cases, so tests are run every time the software changes, maintaining the quality of the application even as it evolves and obviating the need for documents. The ongoing benefit of Agile's specification-as-automated-test contrasts with the written specification, which tends to lose its value as soon as the initial feature development is completed.

Culture Change	Traditional	Agile	Benefit
Project Focus	Designing all features at the beginning, building and delivering to the user with no project budget remaining	Build the smallest feature that will be used, get it in the hands of the user, then build the next feature	Reduction of unused features; best value for the money spent; even "challenged" projects more likely to have use
Measurement	Project plan tracking	Demonstration of working software	Return on investment throughout the project instead of at the end
Subject Matter Involvement	At the beginning for specifications and at the end for delivery	Regularly, at least every other week	Features match subject matter experts' needs, misunderstandings identified early
Requirements Documentation	Large document detailing all thoughts, traceability matrix to verify in testing all requirements are met	Screen mock ups, requirements written as continuously running tests	Improved understanding about what will be built by all parties, requirements that never get out of date and ensure quality throughout project
Design decisions	Made by the vendor during build phase	Made collaboratively during demonstrations	Software better matches users' needs
Quality	Measured at end	Measured every week	Reduction in large architectural changes needed near end of the project; delivery on time
Relationship	Vendor and organization have independent success criteria	Partnership, collaboration, goals aligned	Better software, better time working on the project

Contract Vehicles

Important aspects of contracts when using Agile

Agile development can be done under any contract vehicle, but some contract vehicles make it significantly easier for the team to work together. The two critical aspects of the contract include a mechanism to reprioritize feature requests and a mechanism to specify how much work the team will deliver.

Mechanism to allow for feature reprioritization

In some contracts, each and every desired feature is enumerated. The biggest hurdle comes when the customer wants to prioritize a feature that was not in the original list because it provides more business value than the items that are in the list. This requires a contract modification and often creates friction between the customer and the vendor because their objectives are not aligned.

In the most Agile-friendly contracts, the desired features are in an appendix or not specifically enumerated in the contract but enumerated in a work plan developed at the beginning of each release. This allows the customer to reprioritize features to meet the most up-to-date business need. The contract still contains in general terms the business value the organization will receive, so it is clear whether the project team is meeting the needs of the customer.

Any contract that specifies exact features will require frequent modifications to get the highest business value. If a contract with features specified as an addendum that can change is unacceptable, one way to minimize modifications is to specify two-thirds of the features and leave the remaining third for discovery. It will still require modifications on any of the specified items, but will minimize the changes due to discovery.

It is important with any contract structure that does not fix the exact feature set that the customer includes people on the project team with domain knowledge. These project team members should have skills to articulate the business value that will be gained with each feature and make priority decisions on the features with the most value. A guiding strategic document or project charter is generally used to help the team make the best decisions.

Generally, Agile contracts also contain provisions that stipulate that the customer can decide, at the end of each demonstration, whether the new software features provide the business value they desired. The quality of the features will be ensured by the definition of the exit criteria before the feature is ever developed, so this is more about whether additional funds should be expended in this part of the software or elsewhere. Thus, the customer will have continual opportunity to assess the effectiveness of developed software, and when necessary either reprioritize what the development team is doing, or decide to end the project if the cost of any additional features would not be a good return on investment. This helps promote continual alignment between the customer and the vendor.

Mechanism to specify how much work the team will deliver

In Agile, the typical unit of measure is story points. This is like a currency that represents how long a feature will take to complete. It varies based on the team performing the work. The software development team often doesn't deal in dollars or days, but the software project manager or engagement manager should be able to translate this for the customer.

Agile teams estimate features in story points (baseline) and measure how long a feature took to complete (actual) in story points. Agile teams can tell customers what their average number of story points is in a timeframe (velocity).

Time and Material contracts (T&M) are often easiest to use with Agile teams because the features may be enumerated, but no specific timeframe is listed for every feature. Indefinite Delivery Indefinite Quantity (IDIQ) is also very easy to work with when the Agile team has a set release schedule. Both allow the customer and development team to work together to determine how elaborate each feature needs to be. If there are timelines that need to be met, the customer can front-load the contract. Based on the feedback loops, contractor performance is easily measured. Unfortunately, these contract vehicles are out of favor, so other contract vehicles need to be examined.

A similarly easy contract to work with is a Firm Fixed Price, Level of Effort contract (FFP LOE), as long as the scope of work is defined either as the number of story points or the team size that is producing the work. This gives the customer similar benefits to determine how elaborate each feature needs to be and the ability to increase the level of effort (have more points delivered) if there is an urgent business need.

A Firm Fixed Price contract (FFP) that defines the scope of work as a static number of story points can work, but it does not give the customer the flexibility to accelerate the work if they need to without a contract modification. In our experience, new legislative requirements or political landscape changes drive the need to accelerate functionality in the software or add entirely new features based on emerging needs.

Some contracts have made the period of performance of a contract mirror the proposed software releases. This reinforces the idea of evaluating each release (in addition to each demonstration) for sufficient value and examining whether the organization would want to continue with the development of further features. Sufficient planning for deployment and post release support needs to be considered if this avenue is chosen.

For those organizations that cannot avoid enumerating features in the contract, a split contract is recommended. The first part of the contract is to define the basic requirements and should place heavy emphasis on paper prototyping and expected use cases. This will help uncover domain experts' assumptions and contractors' gaps in understanding. The prototyping and requirements inventory should not represent more than ~5% of what the organization intends to spend on the project. The

second phase will include the development, testing and deployment. Generally option periods will be equivalent to the second phase.

Regardless of which contract vehicle that is used, contract elements need to be added that ensure agile principles are adhered to. Deliverables should focus on demonstrations of working software rather than an updated project plan. Periodic delivery of automated test results should also feature prominently in the deliverables.

Contract Change	Traditional	Agile	Benefit
Features	All features enumerated in contract	Business goals enumerated in contract, features enumerated at the beginning of each release in a work plan	Fewer contract modifications with same or better level of control on feature development
Unit of work	Hours of project team (T&M) or a list of features (FFP)	Story/Feature points (T&M, FFP LOE, or FFP)	Allows the flexibility to change the features to meet the business needs while keeping the amount of work agreed to constant; fewer contract modifications
Period of performance	Typically annual with features split out by year	Coinciding with release schedule	Natural time for evaluation, doesn't compete with busiest time of contracting year; fewer modifications related to features moving to a new POP
Deliverables	Project plan, requirements document, testing plan, traceability matrix, software at end	Regular demonstrations of working software throughout, mock ups for designs, automated tests for quality checks	Higher quality software, on time; value throughout project instead of at end
Assessment of contract fulfillment by Contracting Officer	Reading through requirements documents and trying to reconcile them to the product produced	Attend a release demonstration or project team meeting	Less conflict; easier assessment on whether contract obligations have been met

Evaluation Factors

Agile experience and prior record of succeeding

Agile is a philosophy of partnership. Customer collaboration is one of the four main principles. Trust is a critical component for a working partnership. Listening skills are also a key success factor. If contractors are seen as "them" instead of an extension of your own team, a paradigm shift may be needed to make Agile development as successful as it can be.

Domain knowledge

Domain knowledge improves shared understanding, which in turn improves both the speed at which software can be developed as well as the usefulness of your final product. An agile vendor who has knowledge of your domain will improve the project's success probability. Frequent feedback loops can overcome some shortcomings of a vendor who does not have knowledge of your domain, but collocation should be considered in those cases to minimize communication gaps. The developer needs to have someone on its team with some domain knowledge to facilitate communication.

Recognized Agile processes

The word Agile is used as an umbrella for a lot of different processes. Some misuse the term to mask an ad hoc process that does not have the level of quality that you should expect with an Agile team. Having a daily stand-up (a meeting where completed work and planned tasking are shared with the entire group) or using the word "Scrum" does not equal Agile.

You should look for the following items in any agile process:

- Iteration length – This is the measure of how long the development team will work between feedback loops with the customer. Iterations should be short. Every other week is a typical duration. Anything longer than one month is concerning as it increases the probability that the software development team and the customer will get out of sync.
- Demonstrations of working software – At the end of each iteration, the customer should expect to see a demonstration of the work that the software team has completed over the course of the iteration. Early in a project, there may be little to demonstrate because there is technical setup or infrastructure needed. By the time the second or third iteration is complete, customers should expect to see some aspects of what they want demonstrated as working software, with an invitation for feedback. These may be very small features (e.g. User is able to log in and get a stub of a home page), but they should be demonstrable nonetheless. They should also be of high quality having undergone testing as though the feature was going to be released.
- Release cycles – How often is software released? One of the large benefits of Agile is that software gets into the hands of the users much earlier in the process, providing immediate return on investment. Somewhere between continuous deployment (immediately after the

feature is completed and tested it is made available to production) and three months is a typical release cycle.[2]

- Velocity over time – How much work is completed by the development team in each iteration? A team that has been doing Agile should be able to communicate a historical trend. Without this measurement, a team will not be able to accurately communicate when the estimated features will be delivered. If a team has a steady decline in velocity, it may indicate that their quality practices are not sufficient for sustainable development.
- Does each feature have automated, machine-repeatable testing by the time it is demonstrated? This may be the most important indicator of the quality of the software and the confidence you can place in the team's ability to deliver when they say they are going to.

Automated tests

Under the exterior, good software consolidates similar functions used across the application into a central location. The benefit of this is that the business logic (the rules by which the software operates) is in a single place and if requirements change in a widely-used process, the code changes required are limited to a single place. The down side of this is that changing one part of the software can break other parts.

Reliably identifying these unexpected breakages (a necessity for agility in significant projects) requires automation. While manual testing is still an important activity in agile development, the goal of test automation is to know when touching one area is impacting another. If the team has automated tests, they can be run with each code change (called continuous integration). This will save a lot of costly rework and defect tracking, because issues will be identified immediately as they are introduced instead of further downstream, when they are costlier to fix.

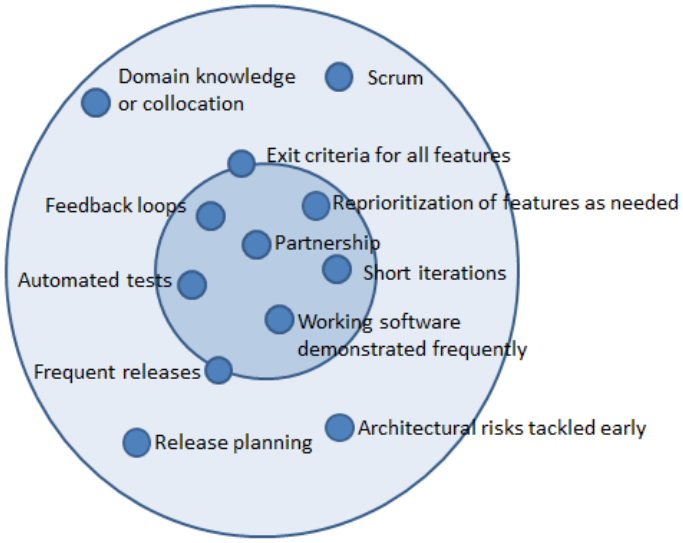
When contracting for software, part of what you will want to ask for is automated unit tests. Unit tests will test each of the individual functions of a feature. Automated integration tests, which string together several unit tests into a business process, are also desirable. Agile teams will be familiar with these and use specialized tools to create them. Automated user interface tests, which test how a user would interact, may or may not provide sufficient return on investment, depending on how many different IT platforms you plan to support.

[2] Release cycles are sometimes impacted by an infrastructure group's inability to release as frequently as the software development team is able to release. This is why agencies often consider going with a SaaS (Software as a Service) or a Hosted model of IT support.

Feedback loops

Lastly, evaluating an Agile company should look at other feedback loops in their process.

- Estimating features – Agile looks to avoid big design up front (BDUF), understanding that business needs will change. A good amount of design work is needed to accurately estimate each feature; this work will have been wasted if the feature is not immediately prioritized for work. To prevent this waste, Agile teams provide estimates as features are requested. Mock-ups or prototypes should be provided alongside the cost estimates, to help the business understand precisely what feature is planned to be completed for the cost. This allows the business to make an informed go/no go decision. Features not selected go to the Product Backlog if there is a belief that they may want to be worked later.
- Exit criteria for features – How do you know when a feature is complete? If you are working with an Agile team, before the feature is demonstrated, there will have been a discussion about how you will both know that the feature is done. The criteria that you agree to are called Exit Criteria. Agreement before the feature is developed minimizes disconnections and avoids scope surprises that expand effort beyond estimates. An excellent Agile team will convert this list of exit criteria to automated tests, ensuring that the criteria continue to be met as the application changes.
- Evaluating architectural choices early to avoid rework - Are the architectural choices sound? How are you testing them? Nothing can delay a project like learning at the end that the last piece of an integration requires a complete rework of the system. This is a function of insufficient testing and feedback along the way. Agile has a quick feedback loop called a "Spike." The team agrees to investigate a technical problem or choice within a small budget and report to the customer on the results before pursuing it further. Agile teams focus on tackling big risks early to avoid negative surprises late in a project.



Closer to the center represent core Agile practices

Evaluating Contractors after Initial Award

If I were to outsource to a development group, how would I evaluate them? I would ask myself the following questions:

Do I feel like I am getting the appropriate return on investment?

- Are the stories that I prioritized being worked in the order that I asked?
- Is the number of story points that we agreed to being completed?
- Is the team suggesting innovative solutions that will improve my business beyond just automating a current process? Is the team providing tradeoffs and alternate solutions that might be a better choice to accomplish my goals?
- Am I receiving working software that is providing me new information for decision making or minimizing work for my team so they can do something more important for the organization?
- Are new features getting into the hands of my users on a frequent basis, so our investment isn't just sitting on a shelf?

Is the level of quality high?

- Are the features ready for release or do they require significantly more testing after demonstration?
- Are the code coverage numbers (% of code under automated tests) what we agreed to or is this work being deferred?
- Am I seeing sufficient requests for technical stories that indicate that technical/architectural issues are being dealt with or is this work being deferred?
- If I am doing any quality assurance testing or user acceptance testing, what is the rate of defects? Are the defects something that was enumerated in the exit criteria? Are the exit criteria being met prior to demonstration?
- What do the error reports from the field look like? How often do I have to patch a release? How often does the patch create more problems?
- What is the satisfaction level of the end users of the software? Do they rate the software as intuitive to navigate? Is it easy for them to accomplish their tasks?
- Is the communication one of partnership? Do I feel in the loop? Am I rarely surprised?
- Are there regular feedback loops? Is better software being produced as a result of these loops?
- Am I getting feedback within a day of discovery of unexpected technical issues or identification of work (required from a feature that I asked for) that will delay our schedule?

Contracting officers may need to work more closely with the project team to evaluate these criteria, but the benefit to the business is profound. We hope this information will help any team looking to adopt Agile.

Envisage Technologies

Envisage is a high tech software company founded in 2001 to automate complex training operations for high liability industries. We create solutions that make our world a safer place. Our clients include military commands, federal law enforcement academies including the U.S. Department of Homeland Security (DHS), and many state law enforcement and public safety organizations.

Ten years ago, we imagined what could happen if federal, state and local public safety agencies could suddenly increase their readiness through more effective training management and resource tracking. Today, through the Acadis® Readiness Suite, our clients are doing just that. The development of the Acadis Readiness Suite fit the Envisage model perfectly. It gave us an opportunity to couple a profound, real-world understanding of key issues with our Agile software development process to deliver a platform that has transformed how training is managed, automated, recorded and institutionalized (a process that was completely ad hoc before we took on the challenge).

Michelle Cole

Michelle Cole joined Envisage in September of 2006, putting years of expanding responsibilities with software development immediately to benefit on the company's behalf. Many of the most important transitions in the recent history of Envisage bear her mark. Under her direction, the company has become more client-focused, more agile in analysis and software delivery, and more adept at delivering unparalleled client service.

Cole is especially skilled in transitioning development teams from traditional waterfall methodology (linear, stage-by-stage development) to agile methodology (which relies heavily on flexibility and adaptability). She coaches leaders that are interested in moving their teams to Agile methods. Cole previously worked in senior-level management roles at software companies including Landata Systems (a division of Stewart Title), Duration Software and iProperty.com. She has directed teams of more than 100 professionals and managed budgets in excess of $12 million.