Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?

One of the first decisions we face for each of our project implementations at Segue is “Which development methodology should we use?” This is a topic that gets a lot of discussion (and often heated debate). If this is not something you’ve worked with before, a definition of development methodology is in order; put very simply, it’s a way of organizing the work of software development. This is **NOT** about a style of project management or a specific technical approach, although you will often hear these terms all thrown together or used interchangeably.

The two basic, most popular methodologies are:

1. *Waterfall:* (ugh, terrible name!), which might be more properly called the “traditional” approach, and
2. *Agile:* a specific type of Rapid Application Development and newer than Waterfall, but not that new, which is often implemented using Scrum.

Both of these are usable, mature methodologies. Having been involved in software development projects for a long time, here are my thoughts on the strengths and weaknesses of each.

The Waterfall Methodology

Waterfall is a linear approach to software development. In this methodology, the sequence of events is something like:

1. Gather and document requirements
2. Design
3. Code and unit test
4. Perform system testing
5. Perform user acceptance testing (UAT)
6. Fix any issues
7. Deliver the finished product

In a **true** Waterfall development project, each of these represents a distinct stage of software development, and each stage generally finishes before the next one can begin. There is also typically a stage gate between each; for example, requirements must be reviewed and approved by the customer before design can begin.

There are good things and bad about the Waterfall approach. On the **positive** side:

- Developers and customers agree on what will be delivered early in the development lifecycle. This makes planning and designing more straightforward.
- Progress is more easily measured, as the full scope of the work is known in advance.
- Throughout the development effort, it's possible for various members of the team to be involved or to continue with other work, depending on the active phase of the project. For example, business analysts can learn about and document what needs to be done, while the developers are working on other projects. Testers can prepare test scripts from requirements documentation while coding is underway.
- Except for reviews, approvals, status meetings, etc., a customer presence is not strictly required after the requirements phase.
- Because design is completed early in the development lifecycle, this approach lends itself to projects where multiple software components must be designed (sometimes in parallel) for integration with external systems.
- Finally, the software can be designed completely and more carefully, based upon a more complete understanding of **all** software deliverables. This provides a better software design with less likelihood of the "piecemeal effect," a development phenomenon that can occur as pieces of code are defined and subsequently added to an application where they may or may not fit well.

Here are some **issues** we have encountered using a pure Waterfall approach:

- One area which almost always falls short is the effectiveness of requirements. Gathering and documenting requirements in a way that is meaningful to a customer is often the most difficult part of software development, in my opinion. Customers are sometimes intimidated by details, and specific details, provided early in the project, are required with this approach. In addition, customers are not always able to visualize an application from a requirements document. Wireframes and mockups can help, but there's no question that most end users have some difficulty putting these elements together with written requirements to arrive at a good picture of what they will be getting.
- Another potential drawback of pure Waterfall development is the possibility that the customer will be dissatisfied with their delivered software product. As all deliverables are based upon documented requirements, a customer may not see what will be delivered until it's almost finished. By that time, changes can be difficult (and costly) to implement.

The Agile Methodology

Agile is an iterative, team-based approach to development. This approach emphasizes the rapid delivery of an application in complete functional components. Rather than creating tasks and schedules, all time is “time-boxed” into phases called “sprints.” Each sprint has a defined duration (usually in weeks) with a running list of deliverables, planned at the start of the sprint. Deliverables are prioritized by business value as determined by the customer. If all planned work for the sprint cannot be completed, work is reprioritized and the information is used for future sprint planning.

As work is completed, it can be reviewed and evaluated by the project team and customer, through daily builds and end-of-sprint demos. Agile relies on a very high level of customer involvement throughout the project, but especially during these reviews.

Some **advantages of the Agile approach** are easy to see:

- The customer has frequent and early opportunities to see the work being delivered, and to make decisions and changes throughout the development project.
- The customer gains a strong sense of ownership by working extensively and directly with the project team throughout the project.
- If time to market for a specific application is a greater concern than releasing a full feature set at initial launch, Agile can more quickly produce a basic version of working software which can be built upon in successive iterations.
- Development is often more user-focused, likely a result of more and frequent direction from the customer.
- For more Agile Development benefits, please see 8 Benefits of Agile Software Development

And, of course, there are **some disadvantages**:

- The very high degree of customer involvement, while great for the project, may present problems for some customers who simply may not have the time or interest for this type of participation.
- Agile works best when members of the development team are completely dedicated to the project.
- Because Agile focuses on time-boxed delivery and frequent reprioritization, it’s possible that some items set for delivery will not be completed within the allotted timeframe. Additional sprints (beyond those initially planned) may be needed, adding to the project cost. In addition, customer involvement often leads to additional features requested throughout the project. Again, this can add to the overall time and cost of the implementation.
- The close working relationships in an Agile project are easiest to manage when the team members are located in the same physical space, which is not always possible. However, there are a variety of ways to handle this issue, such as webcams, collaboration tools, etc.

- The iterative nature of Agile development may lead to a frequent refactoring if the full scope of the system is not considered in the intial architecture and design. Without this refactoring, the system can suffer from a reduction in overall quality. This becomes more pronounced in larger-scale implementations, or with systems that include a high level of integration.

Making the Choice Between Agile and Waterfall

So, how do we choose? First, we change the game a little (which is what most software development organizations do) by defining our own process. At Segue, it's called our ***Process Framework***, and it's a variation on the traditional Waterfall methodology. Our modifications include use of prototyping where possible to provide the customer a better view of their finished product early in the design/development cycle. This helps to improve the team's understanding of requirements and communication with the customer. After the primary framework of the application is completed per high level requirements, we continue to develop and also to reach out to the customer for refinement of requirements. In this way, we strive to be as iterative as possible without compromising our overall system architecture.

We consider the following factors when considering which methodology to use:

ALIGNING PROJECT TRAITS *with* DEVELOPMENT METHODOLOGIES

PROJECT TRAIT/FACTOR	AGILE	PLAN - DRIVEN (WATERFALL)	COMMENTS
CUSTOMER AVAILABILITY	Prefers customer available throughout project.	Requires customer involvement only at milestones.	Customer involvement reduces risk in either model.
SCOPE/ FEATURES	Welcomes changes, but changes come at the expensive of Cost, Schedule, or other Features. Works well when scope is not known in advance.	Works well when scope is known in advance, or when contract terms limit changes.	Change is a reality so we should prefer adaptability where possible. Contract terms sometimes restrict it.
FEATURE PRIORITIZATION	Prioritization by value ensures the most valuable features are implemented first, thus reducing risk of having an unusable product once funding runs out. Funding efficiency is maximized. Decreases risk of complete failure by allowing "partial" success.	"Do everything we agreed on" approach ensures the customer gets everything they asked for; "all or nothing" approach increases risk of failure.	Contract terms may not permit partial success and may require "do everything".
TEAM	Prefers smaller, dedicated teams with a high degree of coordination and synchronization.	Team coordionation/ synchronization is limited to handoff points	Teams that work together work better, but when contracts are issued to different vendors for different aspects of the project, high degrees of synchronization may not work.
FUNDING	Works extremely well with Time & Materials or other non-fixed funding, may increase stress in fixed-price scenarios.	Reduces risk in Firm Fixed Price contracts by getting agreement up-front.	Fixed price is tough when scope is not known in advance, but many government contracts require it.
SUMMARY	Agile is better, where it is feasible.	Plan-Driven may reduce risk in the face of certain constraints in a contract between a vendor and external customer such as the government.	Through educating our customers about the strengths and weaknesses of each model, we hope to steer them towards a more Agile approach. This may require changes to how our customers, particualarly the government, approach software development projects.

The factors above are not equally weighted; each is assessed depending on the individual project and circumstances.

Although we are starting to see mass adoption of various Agile methodologies in the Enterprise (even DoD and Federal agencies), there are still many organizations that are slow to make the change. It is also very common for organization to transition into more of a hybrid Agile approach that combines aspect of both Agile and Waterfall. The Agile Practice Guide was developed specifically to help organization understand and evaluate the use of Agile and hybrid Agile approaches. The Project Management Institute (PMI) that developed the Project Management Body of Knowledge (PMBOK) Guide collaborated with the Agile Alliance to bundle the two guides in one offering to help organizations, managers and leadership increase agility in the development process.

Once we've decided which basic methodology to utilize, we can further refine the process to best fit our project goals. Ultimately, although the way in which we do our work is important, delivering a solid and maintainable product that satisfies our customer is what really counts.