

Agile or No Agile

Introduction

What is an Agile development? What are the pros and cons of Agile? Is it applicable to any projects of, say, any team sizes, structure, project complexity, etc.?

You probably have read some articles or books about Agile, and some of you even have had some experience already. As such, I will focus more on sharing with you as when and how to take advantage of what Agile has to offer, and how to complement Agile with other methodologies so your project has a better chance to be successful. My discussion won't be an academic one rather it will be a practical one, because after all your success is measured by whether you deliver your software applications on time, within budget and with quality.

My first 15+ years of software development experience has all followed some process driven methodologies. Many projects I've participated involved in several hundred members (system engineers, developers, testers, deployment, etc.). While my first experience in Agile process was about 10 years ago, when one of my teams (~15 developers) was working on a brand new software project. At the time we didn't know the term Agile; we just called it rapid development methodology (it sounds great, doesn't it?). It was a mix of success and failure; we did deliver something quicker, but we also have had our fair share of problems. Now, with many more years of experience and a few more "Agile," I will share my own humble view of whether or how to adopt Agile, other methodologies, or a mix of them for your project.

What Is and Is Not Agile?

I don't think Agile is the panacea for every software project development. I would like to start with my view of what "is" and "is not" Agile:

IS

- Delivers a system in many incremental.
- Customer realized partial benefits sooner -- better return on investment (may be).
- Offers opportunities to identify and correct problems earlier in the process.
- Requires high-level commitment from all team members.
- Best for small projects (single team delivering a project in a few iterations).
- Best for projects in which customers are willing and committed to working with you.
- Best for projects with team members who are collocated.

IS NOT

- A quick and loose process.
- Requires no requirement or design process (documentation, review, approval, etc.).
- Requires less unit test or no QA process.
- Requires no change management and version control for development artifacts.
- Requires no trace-ability & audit-ability.

Potential Traps and Things to Consider

Now, with my brief Agile definition laid out, in this section, I will discuss what can be the potential traps and things to consider to avoid them.

- Because of too much emphasis on “speed,” often documents are out of date, or, worse, non-existing, and source codes are uncommented. What if people leave the team or quit the job? This is even more critical if you have offshore teams or utilize outsourced development.
- It’s human nature that under the Agile (shorter time intervals), people have tendency to take shortcuts such as not documenting and keeping records. Even if the project is successful, it may be difficult if not impossible to maintain. It is even more detrimental if some developers (living documents) leave the project while development is still under way.
- Depending on the complexity of an application, the deployment of each iteration release may require conversion tools to be developed and tested, or manual conversion may be required. Regardless, this takes extra effort and may cause unexpected system down time. An obvious example is when the database schema of the new and old releases is incompatible.
- Additional time has to be allocated for “putting out fire” while developers are working on the next iteration, while in the meantime the current one is in service.
- Human factors are as important. People will be burned out or less effective if a project has too many iterations and development activities last for too long.
- A project is not a brand new one rather an enhancement or a maintenance of an existing system. What if there are not enough knowledgeable members left over, or not enough up-to-date documentation to refer to?
- You may want to consider the practice, in which high-level concepts and requirements are kept in documents while low-level details are explained in source code comments. This way, the documents are easier to maintain because they won’t get out of date quickly, and developers don’t have to go to multiple places to keep things up to date.
- Consider Agile, if you can break a project into many easy-to-manage intermediate deliverables to take advantage of the benefits Agile has to offers such as iterative fine tuning, earlier usage of implemented functions (maximizing ROI). Still, my experience tells me it will be more effective if certain best practices are followed to provide necessary check and balance, and further to avoid the traps mentioned above. If a project is too complex, you may want to consider something like waterfall model for a complete project plan before breaking it down to iterations. Without a “whole” picture, the likelihood of re-architect your software at the iteration level is high. This, of course, is highly undesirable.
- You may want to consider tools, which provide the flexibility for you to customize a process to best fit your situation and enforce your best practices. Productivity enhancement tools are also important, because small time saving become big one due to the repetitive and iterative nature of Agile.
- It will pay off if you mandate certain critical pieces and modules to be reviewed. Not only many heads are better than one, but also more people are knowledgeable of critical pieces. It, too, is true that developers will do a better and more complete job, because they know others are looking over their shoulders.

In summary, Agile is not so much as a methodology rather a set of best practices and processes. It's not an exact science rather a mix of art and science. As such, there are no any hard-and-fast rules as far as a methodology that a project should adopt. What process to use depends on many factors such as, but not limited to, the nature of the project (complexity, management's and end users' expectations, etc.), the structure of the team (skill sets, communication means, commitment, etc.).

Finally, since Easy! Software is an ALM (Application Lifecycle Management) solution provider, I will take this opportunity to mention some of the ALM capabilities, which may help your Agile endeavor, as the closing of this article.

How ALM Tools Can Help Your Agile Endeavor

1. Flexibility to customize a desired development process, which helps you enforcing the best practices. As you gain more experience on Agile, you no doubt will want to finetune your development process. As such, you should consider tool, which can meet that requirement. It will be even more desirable if the tool can handle multiple methodologies for multiple projects so you gain additional benefit such as being able to reassign people freely as a project load increases or decreases without worrying about the cost and time lost to train your people for another tool.
2. Empower efficient and effective individuals (business analysts, developers, managers, etc.) of the team. This is not unique in Agile. Rather, they are even more critical in Agile, because the time interval of an iteration is relatively short. As such, productivity enhancement tools, especially those routinely used by most team members, are highly recommended.
3. Team collaboration is equally important. While individuals are working as efficiently and effectively as they can, they need to be well informed of any relevant activities performed by their peers. To avoid information overload, the information shall be: (i) only disseminated to the ones who have the need to know, and (ii) available and easy accessible for others who have privilege to know.
4. Time management feature. I believe the most accurate and effective way of time reporting (team members submit estimated and actual efforts) is when the person is right on the issue. For instance, if a developer has just resolved a bug and closed that (bug) ticket, it will be ideal if he can record the effort he has spent right when his memory is still fresh. So long as the data are recorded accurately and stored, team leaders can query the data and perform any kind of time management as they deem necessary.
5. Feature based development support. Normally, an Agile iteration will consist of a few prioritized features. As such, a tool not only should keep track of individual changes, but also should be able to track all changes related to a feature. Moreover, it should be able to show the dependencies among changes. An individual developer may only care about his/her assignment, but a leader shall be able to keep track of the status of a feature or features at any time, and, if necessary, to easily drill down to any lower levels.
6. Best practices guided yet easy to use version control tool. Team members shall not be burdened to learn the ins and outs of a version control tool in order to follow the best-practices process the team has set forth. Rather, the tool shall guide them throughout this process. An obvious example is that any changes made will only be incorporated in the release of an iteration when they are associated with an approved change request.

7. Easy to use conflict resolution tool. Conflict occurs when two or more developers changed the same line(s) of a file independently. When it happens, there should be a tool, which can provide visual and easy to edit means so a developer can quickly see the conflict and resolve it.