The Myth of Agile

My first 15+ years of software development experience has all followed some process-driven methodology. Many projects I'd involved have several hundred members (system engineers, developers, testers, deployment personnel). 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, right?). It was a mix of success, we did deliver something quicker, but we also have had our own fair share of problems. Now, with 20+ years of experience and a few more "Agile" ones, I will share my own humble view of whether to adopt Agile, CMM, other methodologies, or a mix of them for your project.

You probably have read some articles or books about Agile process. As such, I don't intend to be an expert to repeat all the benefits and how to go about Agile, rather I will more focus on sharing with you 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 a practical one, because after all your success is measured by whether you deliver your software applications on time, within budget and with quality.

I don't think Agile is the panacea of every software development project. I would like to start with the myth of Agile as highlighted in the table below:

Is	Is Not
Deliver a system in many incremental	A quick and loose process
Customer realized partial benefits sooner. Better return on investment (may be)*	Requires no requirement process (documentation, review, approval, etc.)
Opportunities to identify and correct problems earlier in the process	Requires no design process (documentation, review, approval, etc.)
Best for small projects (single team delivery a project in a few iterations) and internal projects	Requires less unit test or no QA process
Best for projects with team members are co-located.	Requires no change management, source code and version control for development artifacts
	Requires no trace-ability & audit-ability

Potential traps and things to consider:

- Depending on the complexity of an application, the deployment of each incremental release may require conversion tools to be developed and tested, or manual conversion may be required. Regardless whichever, this takes extra effort and may cause unexpected system down time. An obvious example is that 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 release, and in the same time, the current one is cut to service. This is more the case if the team were under extreme pressure to deliver the current production release. Haste-makes-waste rule do apply.
- It's human nature that under the Agile process (shorter time intervals), people have tendency to take shortcuts such as not documenting and keeping records. Even 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.
- 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 outsourced development.
- Human factors are as important. People will be burned out or less effective if a project has too many iterations.
- 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.
- 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 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 advantages of the benefits Agile offers such as iterative fine tuning, earlier usage of implemented functions (maximizing ROI). Still, I feel 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. It, 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 modules are reviewed. Not only more 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, there are no hard and fast rules but the best process for a project. What process to use depends on many factors such as the nature of the project (complexity, management's and end users' expectations, etc.), the construct of the team (skill sets, communication means, commitment, etc.). As mentioned in the beginning, the intention of this article is to stir up more discussions, and, hopefully, via your participation we can all benefit from each other's knowledge and experience.

About the Author

Mr. John J. Hsieh has more than 20 years of experience in software development. He worked at Bell Labs for ten years where he involved in development of many large-scale and mission-critical systems. He cofounded CommTech Corp., which developed software systems for telecom applications. In CommTech he led a software group with more than 120 software engineers. He is the founder of Easy! Software LLC (www.easysoftwarellc.com). Mr. Hsieh has BS and MS degrees in electronic engineering, and a master degree in business administration. You can reach him at jjh888@easysoftwarellc.com.