

Software Development Processes: A Comparison of The Capability Maturity Model & Agile/Scrum

Jennifer Rawson
jenrawson@gmail.com
June 1, 2008

The software industry has a history riddled with the horror stories of software failure. An article published by the Institute of Electrical and Electronics Engineers (IEEE), “Why Software Fails,” estimates that failed software projects cost \$60 billion to \$70 billion per year in the U.S. alone. (Charette 2005) The IEEE article goes on to state: “In the final analysis, big software failures tend to resemble the worst conceivable airplane crash, where the pilot was inexperienced but exceedingly rash, flew into an ice storm in an untested aircraft, and worked for an airline that gave lip service to safety while cutting back on training and maintenance.” Most experts agree, while the reasons for software failures are complex, they revolve around some combination of the following: poor requirements analysis; an inability to manage changing requirements; poor architecture and system-wide design; and a lack of any software process. Quite tellingly, most experts also agree that these failures are avoidable. As a result, businesses, developers and academics have outlined “best practices” to avoid such catastrophic failures and many software development processes have emerged. Software development processes tend to focus on project management, software architecture, and code quality, and aim to mitigate the common problems encountered in software creation. Two popular and very different software development processes are The Capability Maturity Model (CMM) and Agile Development.

The Capability Maturity Model (CMM) was invented by Watts S. Humphrey and introduced in his book, Managing the Software Process, where he describes a framework for evaluating and improving software development processes. Further developed by the Software Engineering Institute (SEI) at Carnegie Mellon University, the CMM emerged from an academic setting and promotes a highly regimented approach to software development. It was originally created to help governmental agencies assess software contractors and has seen widest acceptance in the defense and aerospace industries.

At the heart of the CMM is the idea that “the entire software task [is] a process that can be controlled, measured and improved.” (Humphrey 1989, 4) Humphrey claims that the CMM does not prescribe a specific development process, but instead gives guidance on how an organization can improve existing processes. Humphrey’s model outlines a waterfall

method of software development where each development phase- requirements analysis, architecture, design, coding, testing and documentation- is carried to completion before the next is begun. This is one shortcoming that many critics point out because it is generally accepted now that an iterative approach – one in which each phase of software development is visited many times - is a more realistic approach to software development.

The basic CMM process, which Humphrey calls “The Software Development Planning Cycle,” begins with gathering requirements and ends with the delivery of the product. The cycle happens in this order: requirements are gathered, commitments and requirements are negotiated with the client, the requirements are decomposed and a “Work Breakdown Structure” is created, product size and project resources are estimated, and a schedule is produced. Finally, the plan is presented to the client. If the plan meets the needs of the client then the client signs off and development commences. Future changes are not allowed unless the client is willing to renegotiate the terms, in which case, the process starts over with negotiating the commitment and requirements. (Humphrey 1989, 85-86) The purpose of the strict planning cycle is to avoid budget and time overruns and make sure that the expectations between developers and clients are clear. However, this strict cycle is also the basis of one of the central criticisms of the CMM, namely, that its rigidity artificially constrains the evolution that software projects naturally follow.

The CMM presents five levels of process maturity that describe organizations in terms how much control they have over their processes. Once an organization determines their level of process maturity, they begin to focus their efforts on improvement. An organization focuses on the level above their current position, which outlines specific goals to be addressed and achieved, and then works on achieving those goals sequentially. While Humphrey believes that attracting talented people to an organization is vital, he warns against “the myth of the super programmer,” that all-talented person who’s work exceeds that of the average software team. Rather than relying on super-programmers, he advises relying on a solid software process, which must be in place to support talented programmers, otherwise organizations will suffer from problems of software quality and productivity. (Humphrey 1989, viii -ix)

The CMM successfully addresses two of the four problems that cause projects to fail and Humphrey’s book has many nuggets of good advice. It incorporates good requirements analysis and uses software architecture to ensure good system-wide planning and design. The CMM emerged at a time when many businesses had no process at all and it succeeded in pushing them in the right direction. The CMM is not iterative (which is generally accepted as essential to any good process) - development is conducted in succeeding phases, a new phase is not begun until a previous phase is complete, and phases are not revisited once

complete. As a result, it does not deal well with changing requirements. Some argue that this aspect of the CMM actually hinders a company's ability to produce good software.

Another popular process is Agile Development. At its core, this process embraces changing project requirements. Where the CMM emerged from academia, Agile emerged from the experiences of seasoned software engineers. The academic origin of the CMM is something that draws a lot of criticism of the process. Whether accurate or not, many believe industry experience to be far more valuable to developing processes than academic research and theory. In contrast to Humphrey's statement that "the entire software task [is] a process that can be controlled, measured and improved," Agile developers view software development as "...chaotic and unrepeatable, requiring constant measurement and control through intelligent monitoring." (Advanced Development Methods 1996, 2) Wikipedia succinctly describes the Agile bias, describing it as "...a reaction against 'heavyweight' methods, as typified by a heavily regulated, regimented, micro-managed use of the waterfall model of development." Indeed this sentiment is echoed throughout Agile literature.

Agile can perhaps best be described by an in depth examination of the "sprint". Agile teams work in short iterations called sprints, which last from one to four weeks. A sprint focuses on implementing a small piece of functionality and includes all parts of the development process: requirements analysis, architecture, design, coding, testing and documentation. At the end of a sprint a completely functional piece of software is delivered. The product will not be ready for market until several sprints have been completed, but by keeping the sprints short, the goals and priorities for the project can be reconsidered often.

These short iterations allow the Agile team to be very flexible. If requirements change, the team readjusts priorities for the next iteration, which is only one to four weeks away. The process is lightweight because Agile does not produce detailed documentation, and while there are guidelines for development, there are few hard and fast rules. Some criticize Agile for not having enough structure, for not allotting enough time for system wide architecture, and for having an unrealistic expectation that every phase of development can be touched upon in one to four weeks. However, proponents of Agile say that the process works, which is being validated by increased acceptance among many leading companies such as Google and Yahoo.

Agile emerged from several software development movements, one of the most important being Scrum. Ken Schwaber and Jeff Sutherland, both accomplished developers and contributors to The Agile Manifesto (see agilemanifesto.org), invented Scrum. Schwaber describes Scrum as "a process for managing product development." (Schwaber 2006) Sutherland says the primary inspiration for Scrum was an article titled "The New New Product Development Game," written by Hirotaka Takeuchi and Ikujiro Nonaka, and

published in the Harvard Business Review in 1986. (Sutherland 2001, 7) The paper describes a process that sheds the traditional sequential approach to product development and embraces a flexible, holistic, and team oriented strategy. Takeuchi and Nonaka describe their strategy as a “rugby” approach, because “In rugby, the ball gets passed within the team as it moves as a unit up the field.” Self-organizing project teams “go the distance as a unit, passing the ball back and forth...Rather than moving in defined, highly structured stages, the process is born out of the team members’ interplay.” The multidisciplinary team, or Scrum, engages in iterative experimentation to reach the project goals. (Takeuchi and Nonaka 1986, 2, 3) One of the brilliant insights presented in this paper is that developers need structure, but not so much that it suffocates creativity.

Takeuchi and Nonaka believed that a new approach to product development was needed to keep businesses competitive, and indeed it inspired a revolution in software development. Jeff Sutherland writes that what intrigued him was “Takeuchi and Nonaka’s description of the team building process in setting up and managing a Scrum. The idea of building a self-empowered team where everyone had the global view of the product on a daily basis seemed like the right idea”. (Sutherland 2001, 3) In a presentation at Google (available as a Google Tech Talk Video), Schwaber explains that Scrum isn’t a complex methodology. It has some basic guidelines but is very simple and loose, assuming that teams are intelligent and trusting people to come up with the best solutions in their circumstances. (Schwaber 2006) Teams are trusted to organize and manage themselves, and embrace the inevitable changes that happen over the course of a project. Developers set their own goals, so responsibility for meeting those goals rests fully on their shoulders. Customers are included as valuable members of the Agile development team. The Agile Manifesto website states “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.”

Agile development deals effectively with the problems that cause software to fail. It includes customers on the development team, promoting continuous communication so that project goals are constantly being articulated and defined in more detail. It embraces changing requirements, allowing stakeholders to alter the project’s goals and requirements at the end of each one to four week sprint. This flexibility ensures that a product will not be obsolete because it can continually adapt to a changing market. Although some criticize the process for not including enough software architecture, it does incorporate some, particularly in early sprints. Also, Scrum requires that whatever a team commits to having done in a sprint, be in fact completely done. “Done” includes testing, documentation and refactoring for code quality. This definition of done ensures that the product will be

developed with high quality code and, with practice using Scrum methods, empowers developers and managers with the information needed to realistically assess time and budgetary requirements for a project.

There are proponents and critics of both Agile/Scrum and the CMM. Regardless of the camp, all experts now agree that an iterative model of software development is superior to a waterfall model. Agile is iterative while the CMM is waterfall, so it can be argued that Agile is superior on that basis. The CMM includes good requirements analysis and incorporates software architecture, areas in which some argue that Agile falls short. However, based on the problems that cause software to fail presented in the introduction, Agile seems to address more of them than the CMM. Ultimately, the true test of each process will be its level of acceptance in industry and its longevity. The CMM has been around a long time, but is declining in popularity, whereas Agile, although still quite new, is gaining popularity rapidly. Regardless of the choice of development process, the software industry is definitely making progress toward better processes, and ultimately, better software.

References

Advanced Development Methods. 1996. Controlled Chaos: Living on the Edge. White Paper

Bach, James. 1994. The Immaturity of the CMM. The American Programmer, Sept.

Charette, Robert N. 2005. Why Software Fails. IEEE Spectrum Online.
<http://spectrum.ieee.org/sep05/1685/2>

Curtis, Bill. 1993. A Mature View of the CMM. TerraQuest Metrics, Inc., Austin, Texas. SEI, Carnegie Mellon University.

Humphrey, Watts S. 1989. Managing the Software Process.

Jeffries, Ron. 2001. What is Extreme Programming? XProgramming.com.

Martin, Robert C. Why XP?

Object Mentor Website. So Why Use Agile?
http://www.objectmentor.com/omSolutions/agile_why.html. Access Date: July, 20, 2008.

Sutherland, Jeff. 2001. Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies. Cutter IT Journal 14:12:5-11.

Takeuchi, Hirotaka, and Ikujiro Nonaka. 1986. The New New Product Development Game. Harvard Business Review, January-February.

Schwaber, Ken. 2006. Scrum et al. Google Tech Talk video.
<http://video.google.com/videoplay?docid=-7230144396191025011>. Access date: July 20, 2008.

Schwaber, Ken. 1996. Scrum Development Process. White Paper

Schwaber, Ken. 1996. Controlled-Chaos Software Development. White Paper.

Software Engineering Institute (SEI) at Carnegie Mellon website. <http://www.sei.cmu.edu/>

White, Sharon A and Cuahtémoc Lemus-Olalde. 1997. The Software Architecture Process, in proceedings of ASME-ETCE 97, The Energy Engineering Symposium of Energy Week' 97, pp. 170-175, Houston TX., Jan. 29 - Feb 2.