The Impact of Agile Processes on Requirements Engineering

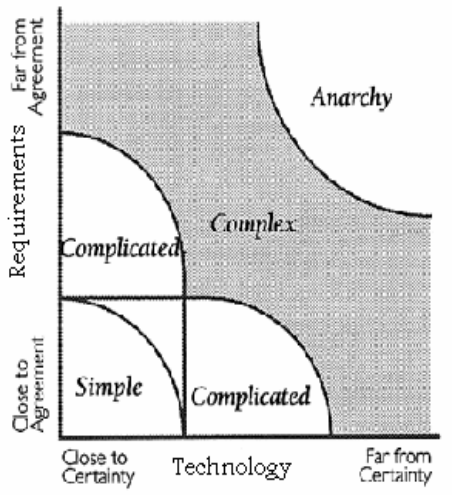
Ken Schwaber
Advanced Development Methods, Inc., AgileAlliance
ken.schwaber@verizon.net

Abstract

This paper discusses how agile processes resolve a paradox of requirements engineering – the need to formulate a clear vision of a system in a world of constantly changing requirements. This paper provides an alternative to current theory based on experiences by agile process practitioners. The paper describes (1) current requirements engineering practices in agile processes, (2) a summary of experiences of practitioners of agile processes regarding requirements engineering, (3) the variance between agile practices and requirements engineering disciplines described in the literature, press, and academia, and, (4) an area of research.

1. Current Agile Practices

Building advanced technology, competitive systems is complicated. As shown in the requirements/technology figure[1], the degree of complexity increases as the requirements are less known and the technology is less certain. When the third dimension of personnel is factored into a project, the degree of complexity for almost any development project is at least complicated and usually complex.



Frequent inspection and immediate adaptation to the results are the primary mechanisms for dealing with project complexity. Various implementations of the following key practices support these mechanisms:

1. Iterative development – frequent iterations generate increments of work that can be inspected to determine the state of the project and serve as a basis for adaptation.
2. Increments of work - composed of working system functionality rather than artifacts. These increments create a 1:1 relationship between progress and product delivery, and provide a mechanism for user feedback to real product rather than arcane internal artifacts.
3. Collaboration – the customers and engineers form teams that work together.
4. Daily meetings – provide a daily picture of the internal status of a project.
5. Adaptation – the teams of developers self-organize daily based on the daily meetings and the developers and customers self-organize at the end of every increment to guide the project to create the greatest value.

[1] Adapted from *Strategic Managment and Organisational Dynamics,The Challenge of Complexity* - 3rd Edition, Ralph D. Stacey , Prentice Hall,Feb. 2000

6. Emergence[2] – the architecture, team structure, and requirements emerge during the course of the project rather than being determined at its outset. The team is guided by preliminary and sketchy visions of requirements and architecture. The architecture is initially elaborated in greater detail for large and complex systems. This simplifies coordinating multiple teams.

2. Agile Experiences

At a recent Agile Workshop[3], the subject of emergence was discussed. The participants indicated that they had successfully applied emergence in literally hundreds of projects over the last seven years. These projects ranged from relatively straightforward systems to n-tier, complex systems applying cutting edge technology to poorly defined business problems. Business, embedded, and life-critical systems had been successfully developed.

The participants felt that the use of refactoring, or periodic cleaning up of business and technical architecture, was required. Otherwise, in all likelihood, the resulting system would be difficult to maintain and sustain. It would contain unrectified designs and code that wasn't orthogonal. However, none of the participants was able to explain why emergence worked. Mike Beedle, Ken Schwaber, and Jim Highsmith referred to works by John Holland and the theory of complex adaptive systems, but even these works only describe emergence in nature rather than providing an underlying theory. Since software development deals purely in logical constructs, the application of any emergence theory will probably be more difficult than with physical systems.

3. Variance Between Formal Requirements Engineering and Agile Processes

The "Call for Papers" for TCRE'02 states that "The fundamental principle underlying requirements engineering is the assumption that a system should be clearly specified before its design and implementation can start. Ideally, a specification should be unambiguous, consistent, concise, traceable, implementation independent, etc., and signed off by all stakeholders involved before any further development is done. ***Failing to do so has in the past caused tremendous cost overruns, delays and many project failures.*** One of the reasons often cited is that the cost and time required for fixing an error increases exponentially as development goes on. These experiences combined with the fact that the most critical decisions are usually made during the early development phases support the assumption that the investment of upfront effort will pay off during the later phases of development."

This paper asserts that the above highlighted assumption is incorrectly based in immature software engineering practices. These practices cause a business and systems model of the envisioned system to be initially built. Costs and delivery dates are predicted from these models. These models are progressively decomposed into further models that eventually result in an operational set of software. Customer requests for changes from these initial models are handled through extensive change control mechanisms. Most of these change control mechanisms require the user to contractually agree that the cost and date may be changed to adopt the changes. The complexity of change orders discourages customers, resulting in projects without many change orders that don't deliver the desired business value. Alternatively, projects become so enveloped with change orders that the system never gets delivered. Even though this model was initially formulated and expressed in waterfall development processes, its underlying focus on top-down development has been continued in other development models until agile processes.

The traditional approach represents a contractually acceptable way for developers to do business with customers, creating a win-lose relationship. This desire for contractual predictability has driven the entire systems development process into the level of detail and bureaucratic overhead that is currently encountered even though "A survey described in this report shows that about 45% of cancelled projects failed due to a lack of requirements engineering effort and that a similar percentage ascribes good requirements engineering as the main reason for project success[4]."

Several other assumptions underlying the above highlighted assertion also need to be more thoroughly questioned. These are that requirements are a valid expression of user needs, that all requirements have equal value, and that the cost of changing requirements becomes more expensive as a project progresses.

[2] Emergence is a key practice in Extreme Programming and Scrum; emergence is supported by DSDM and FDD, although the latter process place more emphasis on initial technical and business architecture design.

[3] OOPSLA 2001, Tampa Florida.

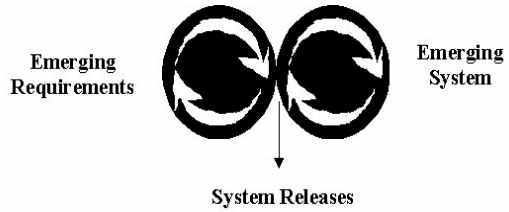
[4] The Standish Group CHAOS report, 1995 and 2001.

One of the Standish group comments was that - of those systems delivered - most systems failed to deliver expected business value. When a project is started, an organization funds it based on expected return on investment. That is, the cost of the project is expected to provide some business benefit or value that is greater than the cost of the project. Requirements engineering is premised on the expectation that this value can be accurately modeled in requirements at the start of the project. Agile processes instead let the requirements that will be provide this value emerge. Emergence is used in concert with increments of working functionality. Most users can only intelligently respond to the system reality rather than such arcane internal artifacts as requirements documents and models. Even use cases fall short of reality. Also, business conditions rapidly change and are often very different at any point in a project from the beginning of the project. Agile processes allow customers requirements to emerge dynamically without penalty to the project. Agile practitioners assert that the ongoing cost of re-engineering a system to adapt to new requirements is less than the cost of architecting a system top-down for out-of-date requirements.

The Dynamic Systems Development Method is an agile process premised on the statement that 80% of a system's value can probably be delivered by 20% of the systems requirements. This assertion certainly is contrary to the requirements engineering practice of ensuring that all requirements are captured. This assertion undercuts the whole practice of requirements traceability. DSDM and other agile processes implement this assertion through frequent collaboration between users and development teams to determine which are the most valuable requirements to develop next in an emerging system of delivered functionality. The focus is on value, not requirements, expressed as:

Business value = f(cost, time, functionality, quality)

Two collaborating cycles of work occur in agile processes. Development teams frequently iterate new increments of functionality. Customers iterate new and prioritize lists of required systems functionality, cost, timetables, and quality based on emerging business conditions. At the end of every iteration, users and development teams collaborate on what to develop next – based on what was just developed and the new business conditions. One possible conclusion might be to create an early release to take advantage of previously unexpected system capabilities or business opportunities.



Traditional requirements engineering practices call for requirements traceability. Initially introduced to map requirements through design to implemented functionality, traceability sprouted into a whole business supported by dictionaries and version control mechanisms. Requirements not only change through agile projects, they evolve and emerge. Since business value is the driver, rather than requirements, the failure to implement one or more earlier stated requirements may simply reflect their diminished business value. The true driver of a project is business value (return on investment), not completeness of requirements implementation.

The last assumption to be inspected is the increased cost of changing requirements as a project progresses. This assertion is certainly true in waterfall development using development tools and environments from the 1960's and 1970's. A key enabler of agile processes is the availability of solid development environments that integrate recent development tools, testing tools, source code management tools, and version control tools into flexible, productive development environments. The informed use of these tools offers the ability to rapidly generate high quality releases of system functionality. Combined with solid engineering practices[5], these tools provide the ability to implement emerging requirements throughout the life of a project at no greater cost than implementing unneeded requirements formulated at the start of the project but never changed.

4. Need for Research

Emergence of requirements in support of business value works and has satisfied customers for over seven years on at least hundreds of projects of all types. This paper states some of the reasons why emergence works and a defined approach to requirements engineering is unneeded and often harmful. However, the underlying

[5] As described in various Extreme Programming books, and encapsulated by engineering practices described by Andy Hunt and Dave Thomas in *The Pragmatic Programmer*, Addison-Wesley, 2000.

theory of why emergence works is not yet formulated. Mike Beedle has a research and development program sponsored by the AgileAlliance (www.agilealliance.org) that is starting to formulate such a theory. I hope that we see results from this program and other research at future sessions of TCRE.