

Agile Processes - Emergence of Essential Systems

Systems emerge when built using agile processes. Through iterative, incremental delivery of functionality, business value emerges every iteration. The architecture and design also emerge iteratively during Scrum and xP projects, a somewhat controversial but nevertheless experientially successful practice. Through emergence, agile processes deliver a product to a customer that maximizes business value. This paper describes how this occurs.

For the purposes of this discussion, I'm going to identify two types of systems. The first is the **envisioned system**, a system as initially foreseen and described by customers to deliver needed business value. The second is the **essential system**, a system with that minimum set of functionality, architecture and design that delivers the envisioned system's business value. **Business value** is defined as capability that provides business advantage for a certain cost delivered by a specified date with adequate quality.

The envisioned system is a starting point for a development project. However, it is not a sufficient or correct description of the system that will deliver the business value. One cause of insufficiency is that the business environment changes during the project. A system that would provide business value at the start of the project often is insufficient by the end of the project. Another reason is communications. When the development team translates the envisioned system into a working set of software, inaccuracies, misunderstandings and lack of knowledge cause inaccuracies.

Still another reason the envisioned system is insufficient is that customers change their minds. Scrum's Uncertainty Principle is, "customers don't know what they want until they see it, and they always reserve the right to change their mind." When the customer sees the envisioned system actually working, they often have different ideas about how this functionality could have best delivered the business value

A desirable objective is to execute a systems development project that (1) focuses and delivers the essential system only, since anything more is extra cost and maintenance, (2) takes into account that the content of the essential system may change during the course of the project because of changes in business conditions, (3) allows the customer to frequently view working functionality, recommend changes, and have changes incorporated into the system as it is built, and (4) delivers business value at the price and cost defined as appropriate by the customer.

Traditional approaches to planning and developing systems 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.

The traditional approach represents a contractually acceptable way for developers to do business with customers. A cost and date can be predicted and contracted. Contractual predictability has driven the entire systems development process into the level of detail and bureaucratic overhead that is currently encountered. It is difficult to build a system that delivers the envisioned business value when this contractual approach is used.

A new, more satisfactory approach to systems development has been described by the agile processes (www.agilealliance.org), and, more specifically, by the Scrum agile process (www.controlchaos.com). This process is detailed in the book, *Agile Software Development with Scrum*, Schwaber and Beedle, Prentice Hall, 2001. This approach is known as “collaborative.” Customers and developers collaborate over the life of the project to deliver an essential system with the business value foreseen in the system vision.

The dissatisfaction with the traditional approach was one vector that caused agile processes to arise. The second vector was the significant improvement in software development environments over the last fifteen years. The extensive availability of libraries, components, and supporting operational environments negates the need to extensively decompose models into code. The code becomes the only valuable model of the essential system.

All agile processes focus on business value. Customers and developers collaborate during the development project to make changes needed to deliver that value. Agile projects are not constrained to an initial contracted model that must be delivered, whether it provides business value or not.

Each agile process implements collaboration differently. Scrum implements collaboration and focuses on business value in an essential system using two complementary, simultaneous cycles. One cycle sustains a dynamic list of prioritized requirements that, when converted into a working system, will create the highest business value. The other cycle consists of development iterations that build system increments to deliver the highest priority business value. These cycles are constructed so that the most appropriate essential system iteratively emerges during iterations.

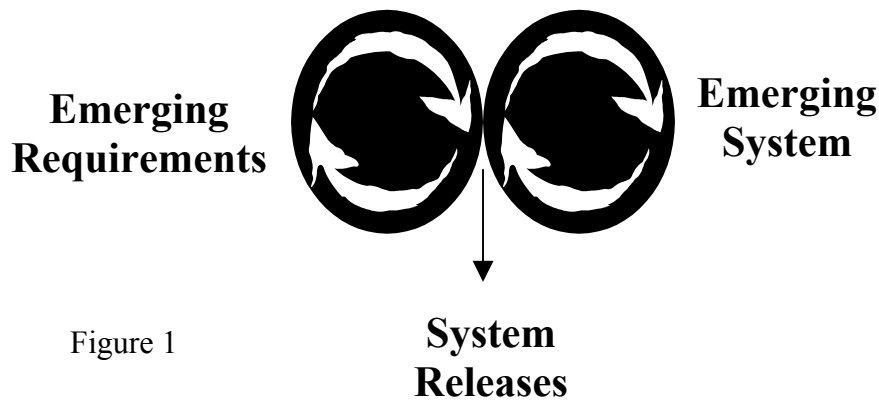


Figure 1

These cycles are implemented through the following steps:

1. A Scrum project starts with the customer's vision of the system. The vision may be vague at first, stated in market terms rather than system terms. The vision will become clearer as the project moves forward. A metaphor of the system is also defined to help guide development and to provide a tangible communication model between customers and developers. An initial vision and metaphor can be usually created in several days, if they aren't already part of the customer's business plan.
2. The customer and development team define requirements that will deliver the highest value business value from the vision. A prioritized list called a Product Backlog is created that consists of these requirements.
3. The development team works for a short, fixed iteration of thirty days (called Sprints) to create an executable increment that contains the top priority business value. The team selects as many requirements as they can build during the Sprint. They only build the architecture and design needed to deliver this functionality.
4. The customer continues defining requirements that will deliver high business value. They are added to the prioritized Product Backlog. The Product Backlog dynamically changes during the project as the business conditions change and through customer response to the product increments created by the development teams.
5. At the end of every Sprint, the customer reviews the working system increment with the development teams to see if it delivers the expected business value, and – if not – what changes need to be made. These changes are added to the Product Backlog and prioritized with the rest of the requirements.
6. When the customer wants to realize the business value achieved to date, he or she will request that product increments built to date be released. One or more Sprints will be used to polish and implement the system into a releasable product.

The customer steers the cost, date, and business value continuously. By increasing the cost, the customer can cause the delivery of business value sooner. By changing priorities in the product backlog, the customer can change the order in which business value is created. By deferring the date, the customer can slow costs. If the customer is dissatisfied with the competence of the developers to understand the business domain and deliver systems that create business value, the customer can terminate the project at any point. The ability to terminate reduces customer risk. Any work completed prior to termination has produced working functionality that may be able to be used by the customer.