Let’s Build a Smarter Method

SDLC 3.0: A Complex Adaptive System of Patterns

Mark Kennaley
Fourth Medium Consulting Inc.
mark@fourth-medium.com

1. Introduction

Recent trends in software development suggest that an appetite is growing for a change from the status quo of *“fads more akin to fashion industry than engineering discipline”*. This seems to be a reaction to the current state of affairs typified by the following disturbing issues:

- Dogma, silver bullets, superstitious declaration, purist interpretation;
- Silo’s, method branching, lack of integration; re-invention;
- Branding , competitive, differentiation;
- Lack of trust inhibiting tacit knowledge transfer.

It is instructive to explore how we arrived at this point. Subsequent to the “accidental waterfall” mis-interpretation of Winston Royce’s 1970 Wescon paper, many attempts have emerged to re-direct software engineering towards the intended path. With the most identifiable roots being Evo or the Spiral Model, two distinctly different and isolated lineages have evolved – the Agile approaches and the Unified Process approaches respectively. Each successive method has emerged from new ideas and experiences, with a focus on rapid value delivery in the Agile case, and risk-value balance in the Unified Process lineage. Unfortunately, a competitive strategy has accompanied these innovations leading to the branding of “complete” methods. Rather than integration, the result has been re-invention due to this isolation. One could rightly question why such a branching anti-pattern (cascading branches) has been allowed to fester for so long, noting that the isolated communities must understand parallel development strategy and ironically embrace continuous integration. Figure 1 illustrates this situation and the need for a re-base of software development method experience.

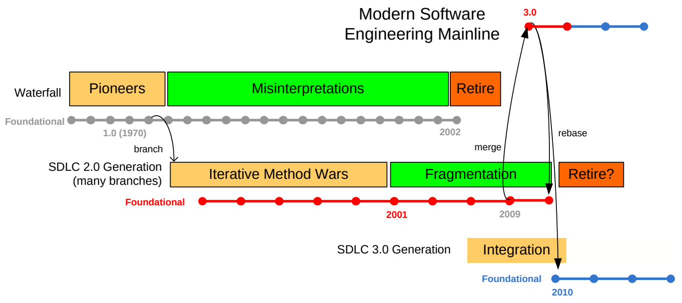

Figure 1. SD Methodology Branching

This is the basis for the label of “SDLC 3.0” [1]. It is time to treat software development experience capture the same as how we treat evolving baselines of a software intensive system.

2. Integrating Modern Software Engineering Methods

The current articulation of methods remains mostly anecdotal and subjective. Additionally, when empirical evidence and supporting models are sought, bodies of knowledge such as Systems Thinking are mentioned only in passing, as in the case of the Agile method *Scrum*. To go beyond this superficial treatment, it is suggested that parallels to the level of study undertaken by the likes of MIT’s System Dynamics Center founded by Jay W. Forrester is required.

If we agree with the observation that software development projects are indeed systems, the first step in applying System’s Theory is to understand the *block diagram* of what has been described as a complex adaptive dynamic system. For this, we need a basis of decomposition – the fundamental building blocks of the system we wish to study. One basis in which modern software engineering approaches can easily be decomposed is through a set of patterns. Such organizational patterns define the essence of the proven solutions to software delivery problems within given contexts. Indeed, others have suggested that the approach to harmonize the various competing branded methods and enable the harvesting of valuable experience is through the decomposition into practices. Through such decomposition, commonality can be identified, and immaterial duplications can be eliminated such that a basis for trust can emerge and the isolated communities can re-engage. Patterns represent a useful level of abstraction above the fundamental elements of a kernel meta-model such as SPEM or ISO 24744. Similarly, patterns seem to be perceived as method community agnostic and therefore highly likely to establish trust with respect to embarking on integrating modern software engineering experience. Unfortunately, the pattern movement that saw moderate success in design and architecture achieved limited momentum in the method engineering problem space. It is contended that this movement should be revived as the basis for harmonization of the fragmented method communities.

Figure 2 that follows represents a domain model of modern software engineering practices. Such a “visual glossary” identifies the synergies where practices are unique within their communities, and where common ground exists that can enable integration into a software engineering “mainline”.

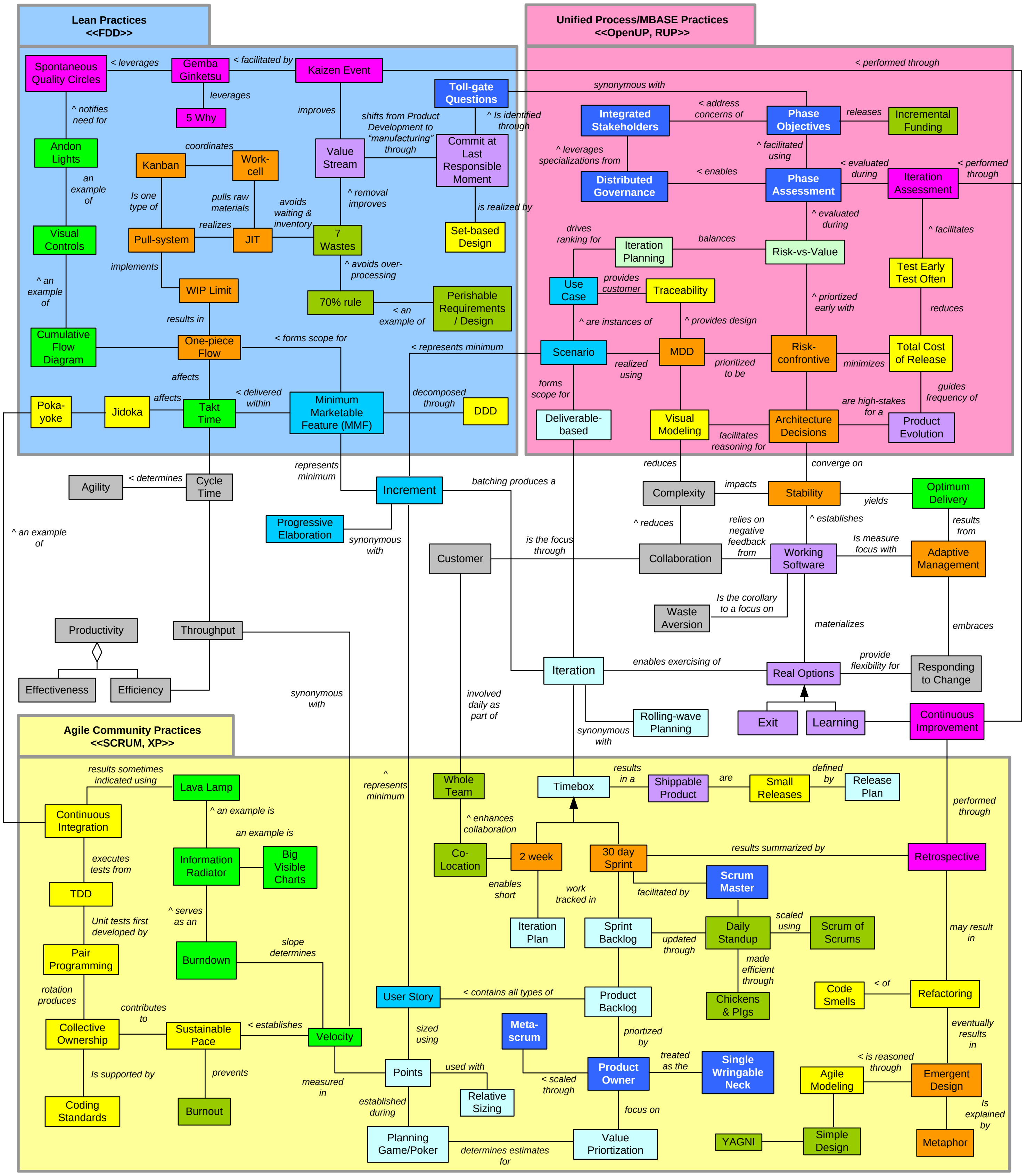

Figure 2. Modern Software Development Integrated through Practices[1]

3. Foundation for Study – Control Systems Engineering

By definition, practices are techniques effective at achieving a desired outcome. Similarly, patterns are solutions to problems in context. The common thread of each of these definitions is they form *tactics* that we employ to influence software development investment outcomes. A cross-disciplinary body of knowledge that applies the same concept for influencing the outcome of dynamic systems is called Control Systems Engineering. It would seem reasonable that we can apply system's analysis techniques from Control Theory to establish a credible basis for when and why we choose one practice over another, and why various practices add value. Yet, Control Systems Theory has received no attention with respect to enabling study of software engineering practices. This is in contrast with other subfields of General Systems Theory like Game Theory, CAS and Agent-based Modeling. Indeed, it is arguable that the foundational study behind Jay W. Forrester's work and the work of his disciples is Control Engineering.

Figure 3 below illustrates the application of a typical negative feedback control configuration to the study of software development practices.

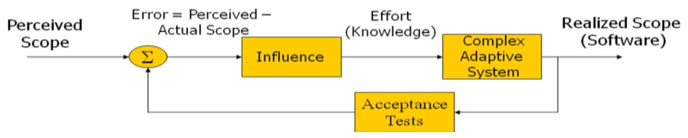

Figure 3. Project Delivery System[1]

The benefits of this approach for grounding the system of practices we need to study is the lengthy history of practical application, and the rigorous suite of analytical and mathematical tools at our disposal. One such mathematical tool leveraged within Classical Control Theory is the Laplace Transform, which is used to enable easier analysis of systems in the frequency domain than with that of the time-domain analysis.

$$\mathbf{L}[f(t)] = F(s) = \int_0^{\infty} f(t)e^{-st}dt$$

Leveraging these tools, it is suggested that the starting point for applying Control Engineering for the study of software development delivery systems is Proportional-Integral-Derivative Control (PID Control) which is the bread-and-butter for industrial process control. Figure 4 illustrates the common configuration.

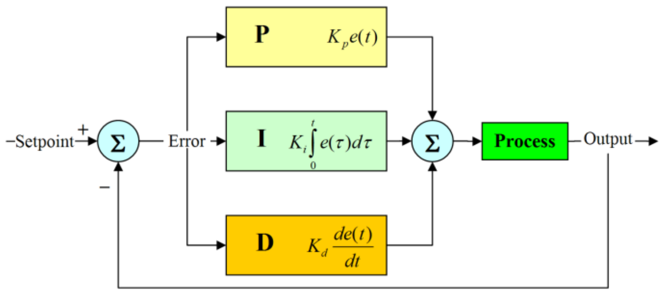

Figure 4. PID Control

Through leveraging the Raleigh curve as a simple approximation of the *plant* (the process block in the previous figure), we arrive at the following transfer function, which is the primary mathematical representation of the system enabling study of the effects of *pole placement* on system dynamics.

$$H(s) = \frac{K(s+z_1)(s+z_2)}{s(s-a)(s-a)}$$

To analyze the dynamic influences of the various practices in a PID control configuration, several classical control theory tools are available. The first is Root Locus Analysis & Design, where we can study such effects as transient response, stability and the influence of various practices on the closed loop transfer function. Figure 5 illustrates this graphical modeling technique:

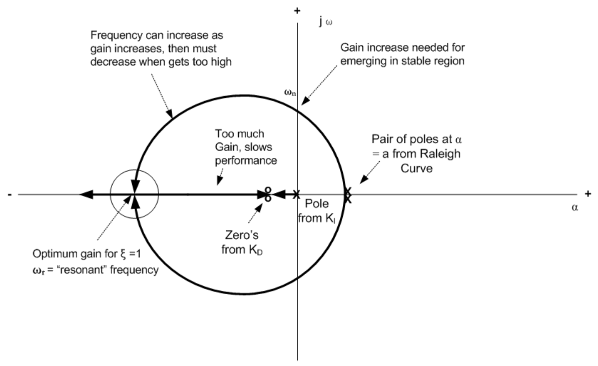
+
j ω
-
+
α
Frequency can increase as gain increases, then must decrease when gets too high
Gain increase needed for emerging in stable region
ω_n
Too much Gain, slows performance
Pair of poles at α = a from Raleigh Curve
Pole from K_I
Zero's from K_D
Optimum gain for ξ =1
ω_r = "resonant" frequency

Figure 5. Root Locus Analysis in s-plane[1]

Similar techniques leveraging the Fourier transform of the system into the frequency domain is Bode Analysis. Through this graphical technique, system robustness through assessment of gain and phase margins are available to assess the result of modifying practice-based control tactics or modifications to the delivery system plant.

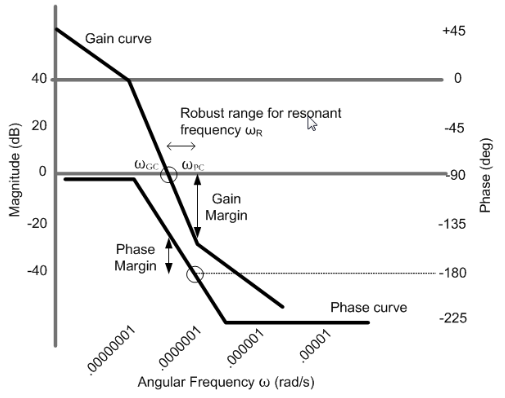

Figure 6. Bode Analysis in frequency domain[1]

Once Classical Control Theory is exhausted, we can move into Modern Control Theory leveraging Linear Algebra and Stochastic Differential Equations. Indeed, Adaptive Non-Linear Control serves as a concrete basis by which we can study the dynamic effects of our practice "control tactics".

References

[1] Mark Kennaley, *"SDLC 3.0: Beyond a Tacit Understanding of Agile"*, Fourth Medium Press, January 2010.