Toward Governance of Emergent Processes and Adaptive Organizations

Peri Tarr, Clay Williams and Brent Hailpern
IBM Research
19 Skyline Drive
Hawthorne, NY 10532 USA
+1 914 784 7278

{tarr, clayw, bth}@us.ibm.com

ABSTRACT
We propose to treat software development processes and software development organizations as *adaptive* and *emergent* entities: ones whose properties are not decidable a priori, but rather, which result from ongoing and continuous response to external stimuli, such as evolving requirements, new enterprise priorities, and changes in available resources. In this paper, we present a view of some important issues that must be addressed to govern such organizations. We provide what we think are interesting starting points for a discussion, as well as a few important questions to address during the workshop.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *software process models, cost estimation, programming teams.*

General Terms
Management

Keywords
Software development governance, emergent software processes and software development organizations

1. INTRODUCTION
Software development governance approaches should align with the organizations and software development activities they seek to govern. But what is the nature of those organizations and activities? For many years, the answer was, they are *designed a priori*: some person, or a group of people, looked at the domain and created an organization and development process that appeared to make sense. Both the organization and the processes were designed to achieve certain key goals. Notable among these are predictability, cost/risk control, and alignment with organizational goals. Indeed, a key motivation for more than a decade of work in the field of software process programming [7] was to improve the predictability and repeatability of software processes [9]. Software development projects that exhibit these characteristics will necessarily reduce the risk to the enterprise significantly and provide a host of benefits that can be leveraged to help achieve the value the enterprise is seeking. Hence, the top-down software development processes that promised these benefits dominated the software engineering field for many years. All of these were predicated on the premise that given a well-defined set of inputs (i.e., requirements), the desired output (i.e., software system) would be produced reliably and predictably.

Decades of experience with top-down, designed software development organizations and processes have demonstrated the fallacy of these beliefs. Schwaber and Beedle [8] write: "Building systems is hard and getting harder. Many projects are cancelled and more fail to deliver expected business value. Statistically, the information technology industry hasn't improved much despite efforts to make it more reliable and predictable. Several studies have found that about two-thirds of projects substantially overrun their estimates [3]." Moreover, most developers hate traditional software processes and methodologies (e.g., [2]). They resent the extra work these processes and methodologies impose on them—work that potentially benefits the process and organization as a whole, but from which they personally derive no benefit and which does not bring them closer to producing the desired software.

The rise of agile methodologies [3]—which share the characteristic that they do not work off a fully specified or even well-defined set of requirements (inputs)—is an interesting and noteworthy response to the failures of top-down processes. Their original proponents were themselves software developers who had experienced the frustration of top-down, designed methodologies. In other words, agile methodologies grew from the bottom up and emerged as best-of-breed practices that *developers* were willing to follow and from which they themselves derived clear benefit, rather than being designed for desired (but unattainable) properties like predictability and repeatability. The agile meta-methodology Scrum [4] [8] took agile processes a step further by empowering—indeed, requiring—software development teams themselves to decide what tools, resources, and methodologies they would use, what goals they will achieve, and who will play particular roles and work on particular tasks. Its premise is that the team accepts responsibility for delivering a software product; therefore, it must have the authority to determine how best to do so.

The relative success of bottom-up, agile software processes and the common failure of top-down, designed software processes has led some (e.g., [1]) to postulate that software development has more in common with complex adaptive systems than with the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SDG'08, May 12, 2008, Leipzig, Germany.
Copyright 2008 ACM 978-1-60558-035-7/08/05...$5.00.

assembly lines after which they were originally organized and run [8]. Although the assembly line view is appealing for the predictability and minimization of risk that it offers—after all, who wouldn't prefer to know that if the requirements are well understood, the same result will always be produced?—the fact is that the requirements on most (if not all) software systems evolve very rapidly in response to myriad external and unpredictable factors, so they cannot be understood a priori. Both the processes and the organizations must *adapt* to these ever-changing requirements and evolutionary pressures. We posit that only *emergent processes*—ones whose definitions emerge in response to these requirements and a set of environmental constraints—and *adaptive organizations* are suited to software development.

Treating software development processes as emergent and its organizations as adaptive necessarily has a profound impact on how we view and address software development governance. If it is not possible to design and manage software development organizations and processes to ensure that they exhibit certain properties, deliver certain value, and understand and manage the risk they pose, how can we govern them?

We believe that software development governance must have a somewhat different focus. Since it cannot design or deterministically manage these emergent processes and adaptive organizations, its goal instead must be to impose a set of criteria that guide the search in the space of possible processes that can emerge. In other words, it becomes a form of environmental pressure on evolution and natural selection. By manipulating a set of parameters, governance attempts to increase the probability that a *good enough* process and organization—ones that are likely to help the enterprise achieve its goals—will emerge.

2. Adaptation and Software Development Governance

Given that governance cannot operate deterministically, several open questions emerge. These include:

- What parameters can you measure and/or change to improve the probability that a good enough process and organization will emerge? What parameters can't you change? What is the space of acceptable values for each parameter?
- How do the parameters affect one another? What tradeoffs are entailed by choosing a particular value for a given parameter?
- What are the feedback loops and mechanisms that are required to ensure an adequate stakeholder/development organization awareness, adaptation, and co-evolution? What is the frequency of feedback required to achieve effective co-evolution?
- How do we populate the feedback loops? What information is useful?

3. Directions for Adaptive, Emergent Governance

In the following subsections, we offer some suggestions to catalyze discussions around these questions. They are not intended to be complete or definitive—rather, they are starting points for a deeper conversation.

3.1 Candidate Parameters for Affecting Emergent, Adaptive Software Development

Rather than concentrating on the design of top-down processes and organizations, the focus of adaptive governance is on defining decision rights, controls, policies, and measurements based around *outcome-aligned* parameters. Experience has shown that process-centric parameters are not good predictors of successful outcomes in software development organizations. We propose a direct focus on outcome that allows organizational structure and processes to emerge.

A common goal of software development is to deliver appropriate functionality on schedule, at or under cost, with the required quality. Ultimately, this goal underlies a larger one, which is to provide value to the enterprise in which the software will be used. Since software development itself is adaptive, the desired outcome is emergent; therefore, we must also dynamically manage evolving risks during development. Thus, candidates for outcome-aligned parameters include:

- Cost
- Time
- Delivered functionality
- Quality
- Value
- Risk

An important question to be discussed is what other outcome-aligned parameters might be useful for implementing software development governance.

3.2 Relationships and Tradeoffs among Parameters

Figure 1: Some Key Parameter Relationships for SDG.

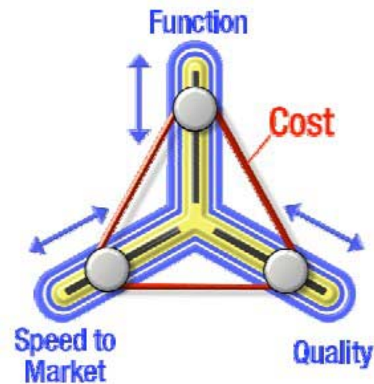

Figure 1 depicts relationships among four key parameters for software development: cost, time, functionality delivered, and quality. As indicated, changing any of these affects the possible set of values that the others can assume. For example, increasing speed to market while maintaining quality and functionality necessitates an increase in cost.

Figure 1 is useful; however, it omits both *value* and *risk*, which are critical parameters of concern for software development

governance. These are interesting, as they are emergent properties that depend on the four shown above.

Two key issues for discussion are approaches to measure these parameters, and identifying interactions between them as their values change.

3.3 Feedback Loops and Mechanisms Enabling Successful Co-Evolution

Figure 2: Strawman Feedback Loops in Adaptive Development

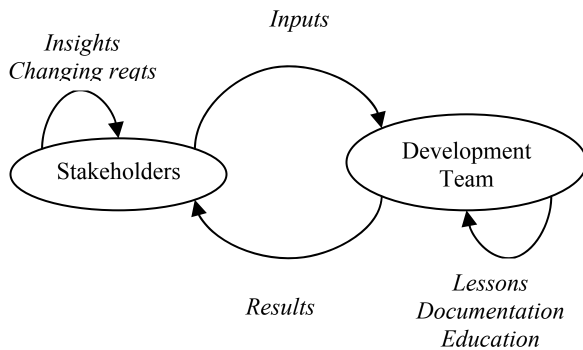

Figure 2 presents a simple schematic illustrating four feedback loops involving the *stakeholders* of a software development project and the *development team* that produces the software. This figure illustrates full feedback between these two populations. The stakeholders provide *input* to the development team regarding their problem domain and their needs as they currently understand them. The development team provides *results* in the form of software. The stakeholders gain *insight* from both the software and reflection on the broader environment in which it will be used. The development team learns *lessons* from their experiences and from interactions with the stakeholders.

To some extent, we see feedback and control occurring in software processes today. We see them most notably in agile processes (especially ones involving Scrum), but they have also been present in top-down processes in the form of evaluations and acceptance tests from customers. We have seen little in the way of understanding these feedback loops, however. Specifically, to govern adaptive development, it is necessary to understand all of the feedback loops that exist, how they interact, how they are affected by various pressures (both internal and external), and how each one can be exploited to maximize the probability that a good result will emerge. As one simple example of this, Schwaber and Beedle [8] noted that all attempts at external control during a Scrum "sprint" result in reduced efficiency of the development team in achieving their primary goals. Therefore, Scrum shields a sprinting team from external control input to whatever extent possible, instead designating specific control points before and after a sprint. Much research is needed to identify, define, understand, and leverage feedback loops.

An interesting question for discussion is what other or alternative models might be useful to represent and affect adaptive development.

3.4 Populating Feedback Loops

Figure 2 provides a high-level view of information flow across two populations. It is important to specify the precise nature of some of this information. Some possible contents of those flows include:

Inputs: Stakeholder priorities, goals, needs, wants, ideas

Results: Software, parameter estimates (including cost, schedule, and function)

Lessons: Development team experiences, knowledge, practices, tasks, tools

Insights: Evolving stakeholder priorities, external observations, knowledge, goals, environmental climate

An interesting question for discussion is what other types of contents would be useful to help the populations co-evolve.

4. Towards an Agenda for Governance Science

This paper has raised a number of questions we believe are critical to be researched and addressed as part of the emerging field of software development government. We believe the agenda for this field should include (but not be limited to):

Evaluate the history: We have hypothesized in this paper that only emergent processes—ones whose definitions emerge in response to these requirements and a set of environmental constraints—and adaptive organizations are suited to software development. A historical retrospective is indicated to evaluate this hypothesis, and to determine which environmental factors and constraints affect governance, and vice versa. One particularly interesting issue is how and why "open" and "closed" development organizations differ governance-wise. Booch's archaeological techniques and studies [5] may provide stories that aid these efforts significantly.

Parameterize governance: We have claimed that governance must act as a form of environmental pressure on evolution and natural selection. A considerable amount of research will be necessary to define the space of dimensions of "pressure points," and to understand the interrelationships among them. Ultimately, we believe that research should look to produce a set of "knobs" that people can turn to effect this form of governance on software development processes and organizations. We need to understand which "knobs" are useful, what values each one can take, the probabilities of affecting the processes and organizations in particular ways with each "knob," and how turning one "knob" affects the ability to turn others. Further, we must devise systems of measurement and feedback to assess the actual results of turning those "knobs" and feeding that knowledge back into the "system," ensuring that it adapts to new knowledge and to the unique environment of each organization and development team.

All of these findings must be evaluated empirically. We hope to see academic studies at universities, as well as industrial and open-source development experiments, where parallel groups approach the same set of tasks using different processes and governance styles and "knobs." Simulations may also prove to be excellent vehicles for understanding and evaluation.

Collecting and analyzing both positive and negative results: In the "hard" sciences, negative results—ones where an underlying hypothesis was not supported by experimental results—are as

important as positive results, because they enlarge the body of knowledge and sometimes result end up overturning long-accepted beliefs. Sadly, it is unusual for computer scientists to appreciate or have the opportunity to publish negative results. We believe strongly that for software development governance to succeed, it must encourage the dissemination of negative results with the positive ones. We cannot afford decades of using methodologies and processes that demonstrably do not work as expected.

5. Conclusions

We have proposed to treat software development processes and software development organizations as adaptive and emergent entities. In this paper, we presented a view of some important issues that must be addressed to govern such organizations. We provided what we think are interesting starting points for a discussion, as well as a few important questions to address during the workshop.

6. REFERENCES

[1] van Aardt, A. 2004. Open source software development as an adaptive system: survival of the fittest? In *Proceedings of the 17th Annual Conference of the National Advisory Committee on Computing Qualifications*.

[2] Atwood, J. 2006. Anything but waterfall. http://www.codinghorror.com/blog/archives/000694.html.

[3] Beck, K. et al. 2001. Manifesto for Agile Software Development. http://agilemanifesto.org/.

[4] Beedle, M. et al. 1998. SCRUM: An extension pattern language for hyperproductive software development. PLOP'98. (Also available at http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P49.pdf.)

[5] Booch, G. 2007. Handbook of Software Architecture. http://www.booch.com/architecture/index.jsp

[6] McConnell, S. 1996. Rapid development. Microsoft Press.

[7] Osterweil, L. 1987. Software processes are software too. In *Proceedings of the 9th international Conference on Software Engineering* (Monterey, California, United States). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 2-13.

[8] Schwaber, K. and Beedle, M. 2001. Agile Software Development with Scrum. Prentice-Hall.

[9] Sutton, S. 2008. Personal communication.