Program Agility and Adaptability

Tyson R. Browning, tyson.browning@incose.org

...agility and adaptability add the most value in the presence of appropriate process standards and planning (and appropriate perspectives on the purposes of both).

"The most flexible process is no process," so admonished a manager at a large aerospace company, and I have heard similar sentiments in the automotive and other industries. The systems engineering and software development communities have experienced a backlash against processes. After all, why would doers of creative, innovative "knowledge work" want to have their actions dictated by rigid directions? Similarly, some program managers question the value of extensive program planning for dynamic programs. For instance, why invest in detailed Gantt charts, showing work schedules in high levels of detail, when those plans will be overcome by events within a few weeks, if not the next day? Sure, they have to show processes to auditors and plans to customers, so they dutifully produce the colorful pictures, but many do not actually find them useful for guiding the numerous tactical, operational, boots-on-the-ground decisions made within the program each day. Outside of some high-level milestones, many engineers and managers seem to prefer to "fly by the seat of their pants." Is this program agility?

What really makes a program more agile and adaptable? Is it just breaking the stifling yoke of processes and plans? Based on my research, the answer, loud and clear, is "No." This article will argue that agility and adaptability add the most value in the presence of appropriate process standards and planning (and appropriate perspectives on the purposes of both). It will provide brief overviews of three research projects that shed light on the path to program agility and adaptability.

Program Standardization, Adaptation, and Value

I previously explored program adaptation and the extent to which it is enabled or inhibited by a standardized program architecture (Browning 2007). In the paper, I proposed that standardization on appropriate architectures is an enabler of adaptability, to a point, past which, it becomes an inhibitor, as depicted in figure 1. The x-axis represents the degree of standardization in a program's architecture (product, process, organization, tools, and goals), and the y-axis represents the value provided by the program. Various value curves such as 1, 2, and 3 may exist depending on the program and situation. As a program moves on the x-axis by becoming more or less structured and standardized, it will either increase or decrease the value of its results, depending on its starting point and the value curve it is on. The proposition is that the value curves are generally concave—the greatest value occurs at a point between the extremes on the x-axis, which is a "sweet spot" between chaos and over-order. Although further research was needed to identify more exact measures of this positioning and the optimum point of maximum value, the paper demonstrated the effect in four case studies of programs with too little or too much standardization.

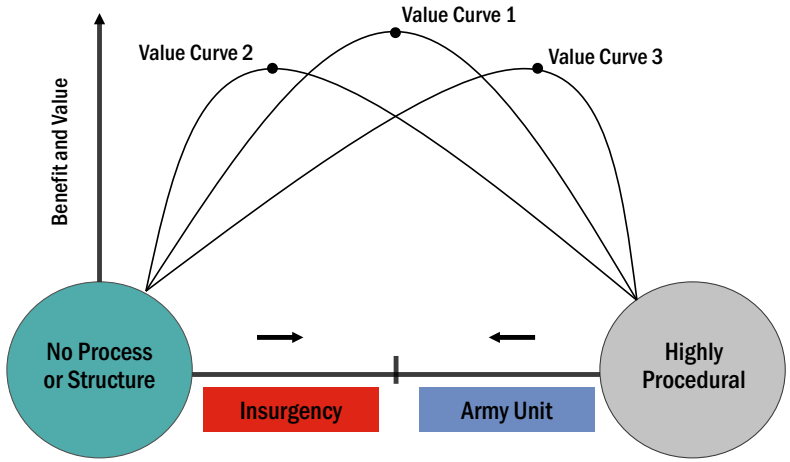

Figure 1. A framework for considering program value as a function of standard structures (Browning 2007)

The first two cases came from Operation Iraqi Freedom in 2003-2004, where a U.S. Army unit and an Iraqi insurgency were involved in what the U.S. Army called "Route Irish," a strategic thoroughfare in Baghdad. The insurgency began as many disconnected cells with a variety of goals, procedures, and tools. They had very little except desire and flexibility. Over time, the cells began to collaborate, sharing knowledge, tools, and procedures; developing communication links and protocols; and coordinating attacks. Although such cooperation and organizational standard-

Table 1: Four cases of program architecture adaptations to increase the value of their results

		Before	Adaptation	After
U.S. Army Route Irish Operations (2003-2004)	**Goals**	Maintain stability in assigned sector, including Route Irish	Increased patrol presence, 24 hour operations on Route Irish	Maintain security and presence on Route Irish in particular
	Product or Service	Relative stability within sector, but increasing attacks on Route Irish	Refined lines of operations, increase in patrols and presence	Attacks decreased immediately
	Process	Maintain presence; set up random traffic control points; counter-mortar patrols; Route Irish presence part of natural patrol route	Increased combat power on route	Removing stopped vehicles; almost 90% of combat power on Route Irish; continuous patrol presence on route
	Organization	Hierarchal, Vertical	Increased information sharing and flow	Increased network connectivity to troop command posts; improved information flow
	Tools	Bradley Fighting Vehicles, M1 Abrams, Up-Armored HUMVEES	Improved systems to defeat improvised explosive devices (IEDs)	Increased capabilities, Warlock systems
Iraqi Insurgency on Route Irish (2003-2004)	**Goals**	Different sub-goals, but general goal of evicting the U.S. military	Recognition that hurting government and civilians caused greater problems for U.S.	Causing instability in the new government (later, sectarian attacks)
	Product or Service	Ineffective attacks	Improved tactics, tools, processes, and organization	Effective attacks
	Process	Simple attacks; no set tactics, techniques, or procedures	The sharing of knowledge and coordination between groups on tool use, tactics, and procedures	Better reconnaissance, supply chain, tactics, techniques, and procedures; more structured
	Organization	Coordination only within a cell	Increased their network, despite different micro goals; used more experts and middle-men	Coordination between cells; deference to shared goals; formal liaisons and information brokers mediating between cells
	Tools	Ineffective small arms attacks	Technological advancements in IEDs	Increase in size and complexity of IEDs
Microsoft Longhorn Program (2005-2006)	**Goals**	Defend the Windows OS and Office Suite	Changed goal to create the next-generation OS more efficiently	Create an innovative, robust new OS
	Product or Service	Extremely complex OS product architecture	Simplified OS product architecture	Simpler OS product architecture
	Process	Highly formal process with nightly builds and error tracing	Restructured software development process	More efficient and effective development process
	Organization	Engineering-oriented program manager; less integrated software engineers writing code	Integrated software developers; brought in new program management and division leadership	Management-oriented program manager; improved integration of program organization
	Tools	Using standard development tools	No major change discovered	Same
Google (2005-2006)	**Goals**	Develop innovative products that change the way the world catalogs and accesses information; private company	Expanded vision of the implications of goal; going public brought additional resources but also additional requirements and constraints	Basically the same, but expanded, and venturing out to other endeavors; public company
	Product or Service	Initial venture was a superior search engine	New products: e.g., Gmail, Google Talk, Earth, Desktop, Maps	Quickly developed products with possibilities for refinement
	Process	Worked independently and very secretively about its processes and finances	No known internal development process changes, but formalized financial reporting processes	Continues to use fairly informal development processes; formal financial disclosure procedures
	Organization	Small groups brainstorming ideas; run by founders; very independent	Hired CEO; partnered with other companies, including some competitors	Continues to use small groups for ideas and product development; in alliances with other companies
	Tools	Massive computer and server infrastructure used for the search engine	Adapting to demand by increasing the size of the infrastructure; likely adopting new tools at the micro level	Larger and still growing infrastructure; micro tool set has likely evolved with the state of the art

ization limited their freedom, the value (to them) of their results improved. The U.S. Army found that its initial procedures, personnel relationships, and tools were not providing their desired values. Giving more autonomy and decision control to local commanders was part of a response that relaxed some organizational standards and led to improved results. Driven by the desire to increase the value of their results, the less-structured program (the insurgency) moved in the direction of increased standardization, while the more-standardized program (the U.S. Army) moved to reduce some prior standards. Table 1 summarizes these changes in terms of their program architectures (product, process, organization, tools, and goals). Although the value curves for each of these programs may have been different—perhaps the insurgency was on value curve 1 in figure 1 while the U.S. Army was on value curve 3—both programs had to move from their original state to a new one to improve their results.

The second two cases looked at rough analogues in the business world, programs at Microsoft and Google around 2006. Microsoft, by then a well-established organization with many formal standards, recognized that its programs had gotten more rigid as the company had grown and developed hierarchical decision-making systems. An earlier focus on innovation had turned to focusing on defense of their market. Like the U.S. Army in Route Irish, Microsoft's program (code named "Longhorn") to develop the next Windows operating system took deliberate steps to increase value by reducing some prior standards. Meanwhile, Google, a relatively new company in 2006, was like the insurgency in Iraq, seeking to increase value by increasing their formal standards and forming partnerships with other entities. Table 1 also summarizes the changes at Microsoft and Google in terms of their program architectures. Many further details of all four of the cases are available in the earlier paper (Browning 2007).

In yet another case of structure enabling agility, at Toyota Motor Company, Spear and Bowen (1999) noted a seeming paradox: "rigid specification is the very thing that makes the flexibility and creativity possible."

Program agility depends on the speed of its adaptation—specifically the speeds of recognition of the need to change, understanding of the objective of change, and making the actual change. Critically, *standards enable speed*. For example, a language such as English has standards and rules of grammar and spelling, but using a standard language does not limit creativity, innovation, and dialogue; rather, it enables it to occur at a higher level of value. Not having an agreed language for communication would decrease the efficiency, effectiveness, and value of communication. At Toyota, processes and rules for the flow of information and materials are highly specified, but these specifications are theory, not law. Toyota can adapt their processes by quickly executing the observe-orient-decide-act loop and applying a scientific method to hypothesize, measure, analyze, and confirm process improvements. I have seen process changes take 6-18 months in some organizations, whereas at Toyota some process changes can occur in less than a day—and then everyone is almost instantly following them as well. Standards are a key enabler of relevant observation, quick orientation, decisiveness, and action.

The earlier paper (Browning 2007) described these changes in terms of a program emergence cycle (figure 2), a kind of closed-loop, feedback control system. A program may attempt to change its architecture in terms of its product, process, organization, tools, or goals. A change in one of these systems usually causes changes in others. A program's ability to change these systems and the rate at which changes are possible both regulate the speed of the change process; both are often affected by the degree of standardization in the program's systems. Short, tight interaction and feedback loops in all program systems facilitate more rapid adaptability, that is, program agility. As changes take effect, new program behaviors and outcomes will emerge, and the cycle continues.

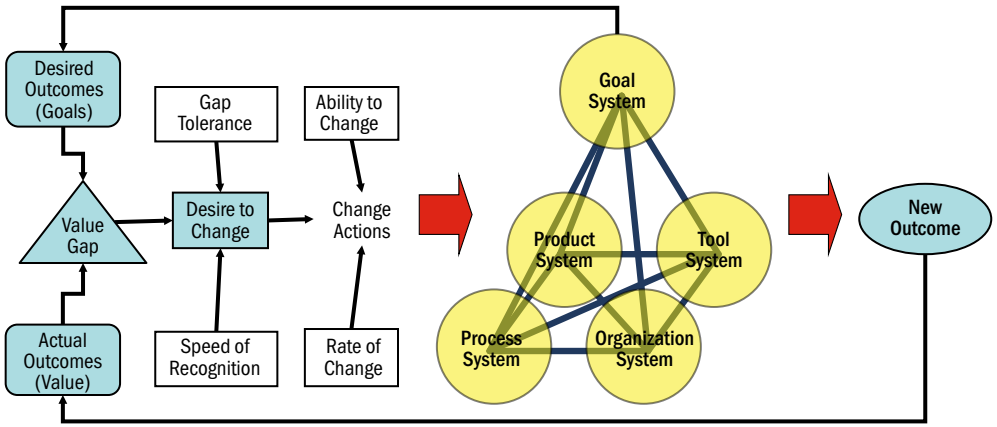

Figure 2. A program emergence cycle, based on changes to program architecture (Browning 2007)

Moving through the cycle quickly (agility) requires that program managers have access to rich, organized, accurate, and integrated information about each of the five systems in the program architecture and the interactions among them. To achieve integration, systems engineers must model each of the five architectures individually and furthermore must map and reconcile the architectures against each other—for example, activities in the process must map to product components, and teams in the organization and tools must map to activities in the process, and all must map to the goals. Hence, program management needs an integrated, architecture-modeling framework for managing a vast amount of program information and pre-testing various improvement options and scenarios. By developing the program's systems as a set of "Lego bricks" that can be rapidly organized towards

changing goals, the standardization and planning required to managing such a program architecture model is an investment in the capability for adaptation. A (product) architecture framework such as the U.S. Department of Defense Architecture Framework (DoDAF) provides a small portion of this capability, for the product system only. A process architecture framework (Browning 2009; 2013) addresses the process system. To date, no one has developed a full architecture framework for a program , although some early efforts to span the five program systems exist (Danilovic and Browning 2007; Eppinger and Browning 2012). Essentially, the need is for something along the lines of "model-based program management" (MBPM) that includes model-based systems engineering of the product system but also model-based planning and analysis of the program's process, organization, tools, and goals.

Adaptive Product Architectures

Two other papers (Engel and Browning 2008; Engel et al. 2012) addressed the specific possibilities of designing products for greater adaptability to increase their lifecycle value. Here the focus is on the product system and its architecture, where architectural patterns such as modularity can increase the ease of adaptability by creating more options for designers and users to redesign, change, and upgrade components and subsystems with lesser disruption and cost to the overall system. This work coined the term *architecture options* for this next-generation method and the associated tools for design for adaptability (DFA). The DFA model defined three metrics—system adaptability factors, component option values, and interface cost factors—to evaluate architecture adaptability during the design phase. For example, greater adaptability can be provided by grouping components with high rates of technological change into modules isolated as much as possible from more stable components via standardized interfaces. Another example of increasing value by appropriate standardization is decreasing the costs of propagating design change across interfaces, by standardizing interfaces.

Adaptive Process Architectures and Program Plans

Lévárdy and Browning (2009) explored the central role in program planning of adaptation in the process architecture, which is the structure of work and interactions that produces the program's deliverables. Programs are temporary allocations of resources commissioned to achieve a desired result. Because each program is unique, the landscape between the current state (the start of the program) and the desired state (the successful end of the program) is often dynamic, uncertain, and ambiguous. Conventional program plans define a set of related activities (a work breakdown structure and activity network) with the assumptions that this set is necessary and sufficient to reach the program's desired result. Popular models for program planning (scheduling and budgeting, for example) and control are also based on a set of program activities that are specified and scheduled a priori. However, these assumptions often do not hold, because, as an attempt to do something novel, only the additional light provided once the work is underway reveals the actual path to a program's desired result.

Hence, Lévárdy and Browning modeled a product development process as a complex adaptive system. Rather than pre-specifying which activities occur and when, it set up (1) a superset of standard activities, each with modes that vary in terms of inputs, duration, cost, and expected benefits and (2) simple rules for activity mode combination. That is, instead of rigidly dictating a specific program schedule a priori, it provided a "primordial soup" of activities and simple rules through which they could self-organize. Instead of attempting to prescribe an optimal process, we simulated thousands of adaptive cases and let the highest-value process emerge. Analyzing these cases led to insights regarding the most likely paths (processes) across the program landscape, the patterns of iteration along the paths, and the paths' costs, durations, risks, and values. Such models can provide decision support needed to move more quickly through the program emergence cycle (figure 2). By assuming during the initial program planning stage that the process will adapt (move away from the original plan), it is possible to improve program planning by understanding the design space of potential paths to success (rather than focusing only on a single one), orient the workforce towards self-organization, and form the capability to cope with many unanticipated situations—in effect, to be more agile. As Eisenhower famously stated, "The plan is nothing; planning is everything." Rather than focus on developing a single, rigid program plan, planning should develop knowledge and understanding about standardized activity options that one can exercise at the latest possible moment to provide the highest value under the current circumstances. Once again, appropriate standardization of the process pieces (rather than a single, overall process plan) creates a basis for the rapid reassembly of those pieces—the capability of agile adaptation.

Conclusion

Programs are complex adaptive systems. They can adapt to increase the value they provide. Their agility depends on the rate at which this adaptation occurs. An appropriate amount of planning and standardization of the program architecture (an integration of the product, process, organization, tool, and goal architectures) enables faster sense and response. However, the requisite planning is seldom funded, the requisite knowledge is seldom well-managed, and the requisite policies

and culture are seldom in place to facilitate an optimum level of adaptability and agility in most large, complex programs. In many programs, the goal of planning is just a plan and not a full understanding of a design space of standardized activity modes and their possible combinations. Many programs push back against their organization's centralized efforts to deploy standard processes, and both programs and functional organizations may not devote sufficient resources to tailoring those standard processes to the program's requirements. Moreover, the development of standard processes in many enterprises was not to support program planning and agile adaptation but rather to attain external certifications, such as ISO or CMM, so the enterprises are hesitant to change them, which makes them adaptable. There are many other examples of resource, knowledge management, policy, and cultural challenges that inhibit appropriate program planning.

This article has presented examples of adaptations via five key systems in program architecture. Most research to date has focused on adaptations in the product and process systems, but the organization, tools, and goal systems matter as well. Consideration for program agility and adaptation ought to occur from the more holistic standpoint of all five of these systems. These challenges present great opportunities for researchers and practitioners in the systems engineering and program management communities. Readers are encouraged to dig deeper into the topics reviewed in this article by consulting the following references.

References

Browning, T. R. 2007. "Program Architecture and Adaptation," Symposium on Complex Systems Engineering, Santa Monica, US-CA, 11-12, January.

Browning, T. R. 2009. "The Many Views of a Process: Towards a Process Architecture Framework for Product Development Processes," *Systems Engineering* 12(1): 69-90.

Browning, T. R. 2013. "Managing Complex Project Process Models with a Process Architecture Framework," *International Journal of Project Management* 32(2): 229-241.

Danilovic, M. and T. R. Browning. 2007. "Managing Complex Product Development Projects with Design Structure Matrices and Domain Mapping Matrices," *International Journal of Project Management* 25(3): 300-314.

Engel, A. and T. R. Browning. 2008. "Designing Systems for Adaptability by Means of Architecture Options," *Systems Engineering* 11(2): 125-146.

Engel, A., Y. Reich, T. R. Browning and D. M. Schmidt. 2012. "Optimizing System Architecture for Adaptability," *International Design Conference (DESIGN 2012)*, Dubrovnik, HR, 21-24, May.

Eppinger, S. D. and T. R. Browning. 2012. *Design Structure Matrix Methods and Applications*, Cambridge, US-MA: MIT Press.

Lévárdy, V. and T. R. Browning. 2009. "An Adaptive Process Model to Support Product Development Project Management," *IEEE Transactions on Engineering Management* 56(4): 600-620.

Spear, S. and H. K. Bowen. 1999. "Decoding the DNA of the Toyota Production System," *Harvard Business Review* 77(5): 97-106.