MIT Sloan School of Management

MIT Sloan School Working Paper 5198-17

Agile for Everyone Else: Using Triggers and Checks to Create Agility Outside of Software Development

James Repenning, Don Kieffer, and Nelson Repenning

June 2017

Agile for Everyone Else

Using Triggers and Checks to Create Agility Outside of Software Development

James Repenning
ShiftGear Consulting
james@shiftgear.work

Don Kieffer

MIT Sloan School of Management ShiftGear Consulting

<don@shiftgear.work>

Nelson Repenning

MIT Sloan School of Management

nelson@mit.edu

June 2017

Version 1.0

1.0 Introduction

You can hardly pick up a business-oriented publication without reading about the ever-increasing pace of change in both technologies and markets and the consequent need for ever-more adaptable organizations. Given the imperative of adaptability, it is not surprising that few words have received more attention in recent conversations about management and leadership than "agile." A quick trip to the Web reveals multiple manifestations, including "Agile Principles," "Enterprise Agile," "Agile Organizations," and a variety of suggestions, including the charge to "Apply agile development to every aspect of business."[1] Organizations ranging from the mighty GE to tiny startups are trying to be both flexible and fast in the ways that they react to new technology and changing market conditions.[2]

The word "agile" appears to have been first applied to thinking about work processes by 17 software developers who met on a ski trip in February 2001.[3] Having experimented with more iterative, less process-laden approaches to creating software for several decades, the group codified their experience in an agile manifesto. "We are…," they wrote, "… uncovering better ways of developing software by doing it and helping others do it." In software development, agile now has a variety of manifestations, including Scrum, Extreme Programming, and Feature-Driven Development.[4] And, the results have been significant. A variety of studies show that agile methods can generate a significant improvement over their more traditional predecessors.[5]

But, what does this mean outside of software? Can agile methods be successfully applied to other types of work? Many proponents (most of whom started in the software industry) argue that the answer is "yes," and a growing collection of books, papers, and blog posts suggests how it might

[1] Goldstein, "Apply Agile Development to Every Aspect of Business."
[2] Leonard and Clough, "How GE Exorcised the Ghost of Jack Welch to Become a 124-Year-Old Startup."
[3] See their manifesto here http://agilemanifesto.org.
[4] See Glaiel, Moulton, and Madnick, "Agile Project Dynamics" for a summary.
[5] See Stettina and Hörz, "Agile Portfolio Management"; Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time*; and West and Grant, "Agile Development: Mainstream. Adoption Has Changed Agility."

be done.[6] The evidence, however, is mixed (and limited). Some authors report success, while others report significant challenges. The blogosphere is also replete with forecasts of the coming agile backlash.

From Practices to Principles and Back Again

Efforts to move agile from the software arena to other domains represent a classic case of the confusion between *practices* and *principles*. Agile methods such as scrum are composed of a set of practices (such as daily progress meetings), and when these practices work, they do so because they help people engage in the work of developing software in ways that capitalize on their natural strengths and offset their inherent limitations. Those strengths and limitations represent a set of principles about the nature of humans working together to accomplish a task. When agile methods work, they do so because the associated practices manifest key behavioral principles in the context of software development. But, as successful as those practices can be when developing software, there is no guarantee that they will work in other contexts. The key to transferring a set of practices from one domain to another is to first understand *why* they work and then to modify those practices in ways that both match the new context and preserve the underlying principles.

The goal of this paper is to help you transfer the insights from software development into the work of your organization and, in doing so, create processes that are more flexible and adaptable. We begin by briefly reviewing the existing theory on flexibility, which suggests that managers face a strict trade-off between organizations that are flexible and those that are efficient. We then show how a feature that is common to both agile methods *and* Toyota's much vaunted production system can significantly weaken that trade-off. Building on that insight, we then offer

[6] Moreira, *Being Agile*.

a framework for understanding agility in any work context. We conclude with examples of using our framework to create processes that are both more agile *and* more efficient.

2.0 Contingency Theory

Perhaps the central notion in the academic theory on organizational design is *contingency*, the idea that organizations and their associated processes need to be designed to match the nature of the work they do. One of the most common variables in contingency theories is the degree of uncertainty in the surrounding environment (often also conceptualized as the need for innovation). When the competitive environment and the associated work are stable and well understood, contingency theory suggests that organizations will do best with highly structured, *mechanistic* designs. In contrast, when facing highly uncertain situations that require ongoing adaptation, contingency theory suggests that organizations will do better with more flexible, *organic* designs[7].

The mechanistic approach to design was first developed by Frederick Winslow Taylor. Taylor's essential insight was simply that if work is regularly repeated, it can also be studied and improved. In stable, well-understood environments, it is thus often best to organize in ways that leverage the efficiency that comes with repetition. As can be observed in any modern factory, in such an environment tasks are precisely specified and the work proceeds serially, moving from one carefully constructed and defined set of activities to the next. There is little need for collaboration in these settings, and the organizational structure that surrounds stable and repeatable work tends to be hierarchical to ensure that everybody follows the prescribed design—operators are watched by supervisors who, in turn, are watched by foremen, etc. The cost of such efficiency is adaptability. Due to the high degree of routinization and formalization,

[7] The classic descriptions of contingency theory can be found in Burns and Stalker, *The Management of Innovation,* and in Lawrence and Lorsch, *Organization and Environment*. An updated summary can be found in several textbooks, including Burton, *Organization Design*.

mechanistic designs are difficult to change in response to new market needs and requirements. Though efficient, a mechanistic design is not agile.

When, however, the environment is unstable and uncertain, work is harder to routinize and therefore organizations cannot rely on precisely designed tasks. As an extreme example, an emergency room physician rarely faces exactly the same situation twice in a given shift. Contingency theory holds that in unpredictable environments like emergency medical care, organizations rely more on things like training and collaboration and less on routinization and careful specification. To manage doctors and ensure high-quality care, hospitals rely on in-depth training and commitment rather than complex hierarchy and extensive standardization. Similarly, the response to a complex emergency can't be organized serially. Rapid and effective responses to unforeseen contingencies require ongoing, real-time collaboration.

Though the contingency notion was first developed more than 50 years ago, its basic insight reappears frequently in the latest management thinking. Many flavors of process-focused improvement, such as Total Quality Management, Six Sigma, and Business Process Re-engineering, are extensions to Taylor's fundamental insight that work that is repeated can also be improved. More recently, the increasingly popular Design Thinking approach can be thought of as a charge to meet ambiguous, uncertain tasks with a more collaborative, less hierarchical design.[8] As shown in Figure 1, the contingency approach gives managers a straightforward approach to designing work: assess the stability of the competitive environment and the resulting work, and then pick the best mix of routinization and collaboration to fit the challenge at hand. If the work being designed is highly precise (making brackets), then it is best to organize it serially, or as we label the cell on the bottom left, using the "factory" mode. Conversely, if the work is highly ambiguous and requires ongoing interaction (treating patients or designing new products),

[8] Brown, *Change by Design*.

then the work is best organized collaboratively, or, as we label the cell on the top right, in "studio" mode.

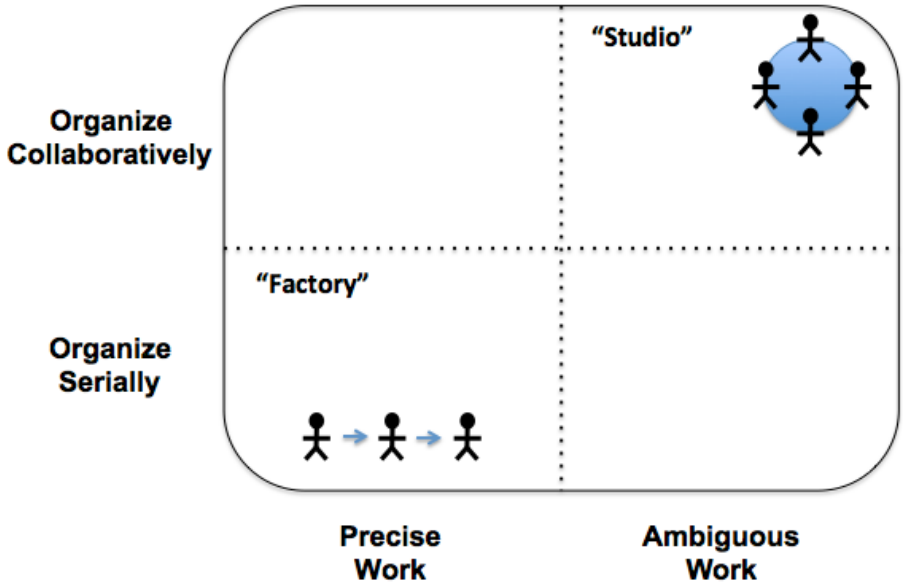

Figure 1: Contingent Process Design

Though powerful, the contingency notion is not entirely satisfying for at least two reasons. First, it describes an unpalatable trade-off: Work done using the factory design isn't very flexible, making it hard to adapt to change in external conditions, and work done using the studio approach often isn't very efficient. Second, when viewed dynamically, few types of work perfectly fit the precise or ambiguous archetype. Even the most routine work has the occasional moment of surprise and, conversely, even the most novel work, for example, designing an entirely new product or service, often requires executing fairly routine analysis and testing activities that support each creative iteration. Academic theory notwithstanding, real work is a constantly evolving mix of routine and uncertainty.

At first glance, agile methods appear to fall more towards the organic side of the spectrum. However, our research suggests a different interpretation. The conventional approach to process and organizational design is almost entirely *static*, implicitly presuming that once a piece of work has been designed, everything will go as planned. In contrast, a *dynamic* approach to design suggests viewing work as an ever-evolving response to the hiccups and shortfalls that are inevitable in real organizations. Viewed from this perspective, agile methods actually transcend the traditional contingency framework by creating a mechanism for moving *between* the two basic ways of organizing work. And, by cycling between factory and studio modes, an agile approach can considerably weaken the trade-off between efficiency and adaptability.

3.0 A Dynamic Approach to Contingency

Understanding the Andon Cord

To introduce a more dynamic approach to contingency, let's begin with a well-known example of work and organizational design, Toyota's famous Andon cord. Work on Toyota assembly lines is the epitome of the serial, mechanistic design. Tasks are precisely specified, often detailing specific arm and hand movements and the time that each action should take. In a plant we visited recently, training for a specific role began with the trainee learning to pick up four bolts at a time, not three and not five. Only when the trainee could pick four regularly was she allowed to learn the next motion. But, despite an attention to detail that would make Taylor proud, sometimes things go awry. In the Toyota scheme, a worker noticing such a defect is supposed to pull the Andon cord (or push a button) to stop the production line and fix the defect.

While the management literature has (correctly) highlighted the importance of allowing employees to stop the line,[9] what happens *after* the cord is pulled might be more important. On a

[9] Liker and Hoseus, *Toyota Culture*.

recent visit to a Toyota supplier in Toyota City, Japan, we spent the morning touring the factory. During our tour, one operator was struggling to complete her task in the allotted time (the work was precisely specified) and hit a yellow button, causing an alarm to sound and a light to flash (this factory has replaced the Andon cord with a yellow button at each operator's station). Within seconds, the line's supervisor arrived and assisted the operator in resolving the issue that was preventing her from following the prescribed process. In less than a minute, the operator, now able to hit her target, returned to her normal routine, and the supervisor went back to other activities.

What, from a work design perspective, happened in this short episode? Take a look at Figure 2. Initially, the operator was working in the factory mode (the box on the lower left), executing precise work to a clear time target. But, when something in that careful design broke down, the operator couldn't complete her task in the allotted time. Once the problem occurred, the operator had two options for responding. She could have found an ad hoc adjustment, a workaround or shortcut that would allow her to keep working. But, as has been documented in several studies, this choice often leads to highly dysfunctional outcomes.[10] Alternatively, as we observed, she could push the button, stop the work, and ask for help. By summoning the supervisor to help, pushing the button temporarily *changed* the work design. The system briefly left the mechanistic, serial mode in favor of a more organic, collaborative approach focused on problem resolution. When the problem was resolved, the operator returned to her normal task and to the serial design. The Toyota production system might first appear to be the ultimate in mechanistic design, but a closer look suggests something far more dynamic. Using the Andon cord, the system actually moves between two modes based on the state of the work. Though the nature of the work couldn't be more different, such movement is also the key to understanding the success of agile.

[10] See, for example, Repenning and Sterman, "Capability Traps and Self-Confirming Attribution Errors in the Dynamics of Process Improvement"; Leveson, "A Systems Approach to Risk Management through Leading Safety Indicators."

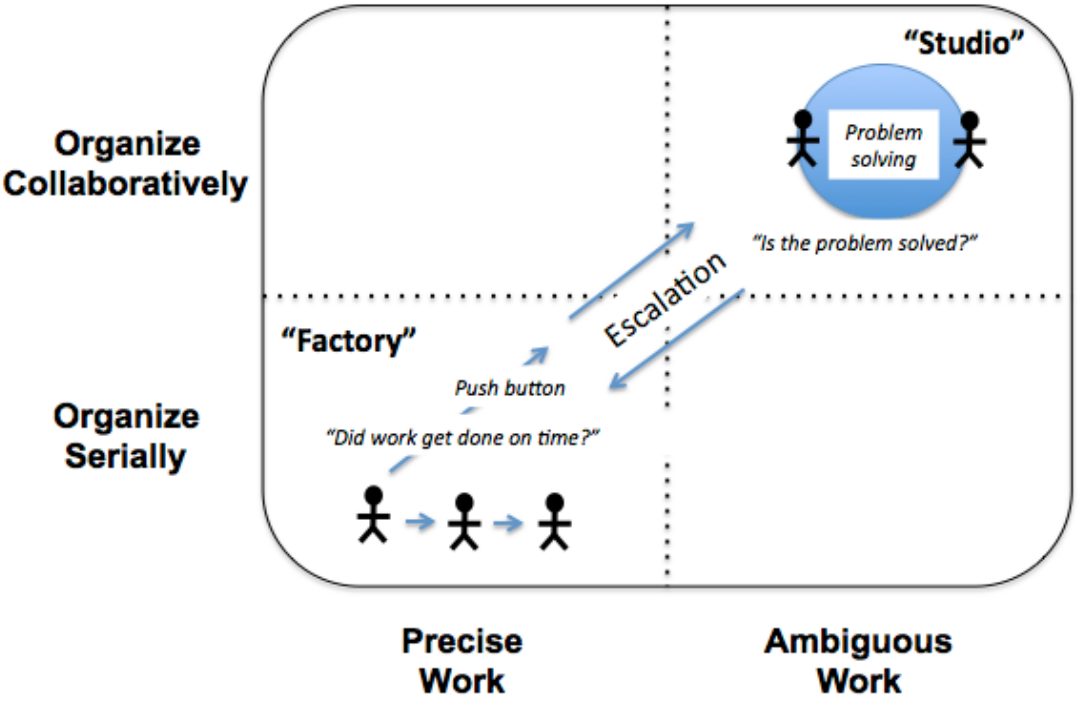

Figure 2: The Andon Cord as Dynamic Design

Understanding Agile

As we discussed in the introduction, the last two decades have witnessed a significant change in the conduct of software development. Whereas software was once largely developed using the waterfall approach, agile methods such as scrum and extreme programming have become increasingly popular. Building on the example of the Andon cord, consider how waterfall and agile approaches differ from a dynamic perspective.

In the waterfall approach, the development cycle is typically divided into a few major phases, based on the type of work that is done in the phase. A typical version includes a requirements phase (figure out what the customer wants), an architecture development phase (map the overall system of modules to deliver those requirements), a detailed coding phase (develop the individual modules), and then a testing and installation phase (make it work). As shown in Figure 3, a waterfall project typically cycles between three basic modes of work. First, the bulk

of the time is spent by architects and software engineers working individually or in small groups, completing whatever the specific phase requires (e.g., requirements, lines of code, tests). Second, typically on a weekly basis, all of those people leave their individual work to come together for a project meeting, where they report on their progress, check to ensure mutual compatibility, and adapt to any changes in direction provided by leadership. Third, at the end of each phase, there is a more significant review, often known as a "Phase Gate Review," in which senior leaders do a detailed check to determine whether or not the project is ready to exit one phase and move to the next phase. Development cycles for other types of non-software projects often work similarly.[11]

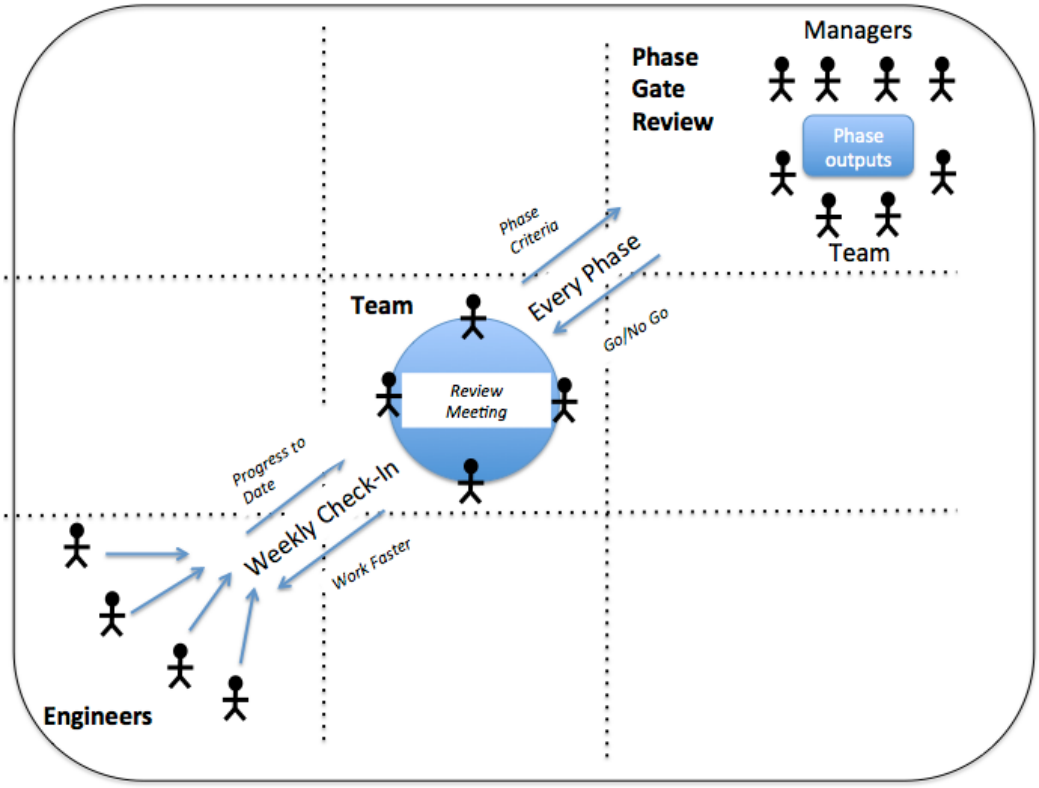

Figure 3: Waterfall Development as Dynamic Design

[11] Ulrich and Eppinger, *Product Design and Development*.

Agile development processes organize the work differently. For example, in the scrum approach[12] (one version of agile), the work is not divided into phases based on the type of work (requirements, coding), but rather into short "sprints" (typically one to four weeks in length) focused on completing all of the work necessary to deliver a small but working piece of software. At the end of each sprint, the customer then tests the new functionality to determine whether or not it meets the specified need.

As shown in Figure 4, the agile approach also has three basic work modes—individual work, team meetings, and customer reviews—but it cycles among them very differently than its waterfall counterpart. First, proponents of agile suggest meeting *daily*—thus moving from individual to teamwork—in the form of a stand-up or scrum meeting, where team members report on the day's progress, their plans for the next day, and perceived impediments to progress. Second, agile recommends that, at the end of each sprint, the team lets the customer test the newly added functionality. Finally, in something akin to the Andon cord, some versions of agile also include immediate escalations to the entire team when a piece of code does not pass the appropriate automated testing, effectively again moving the system from the individual to the team mode.

[12] Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time*.

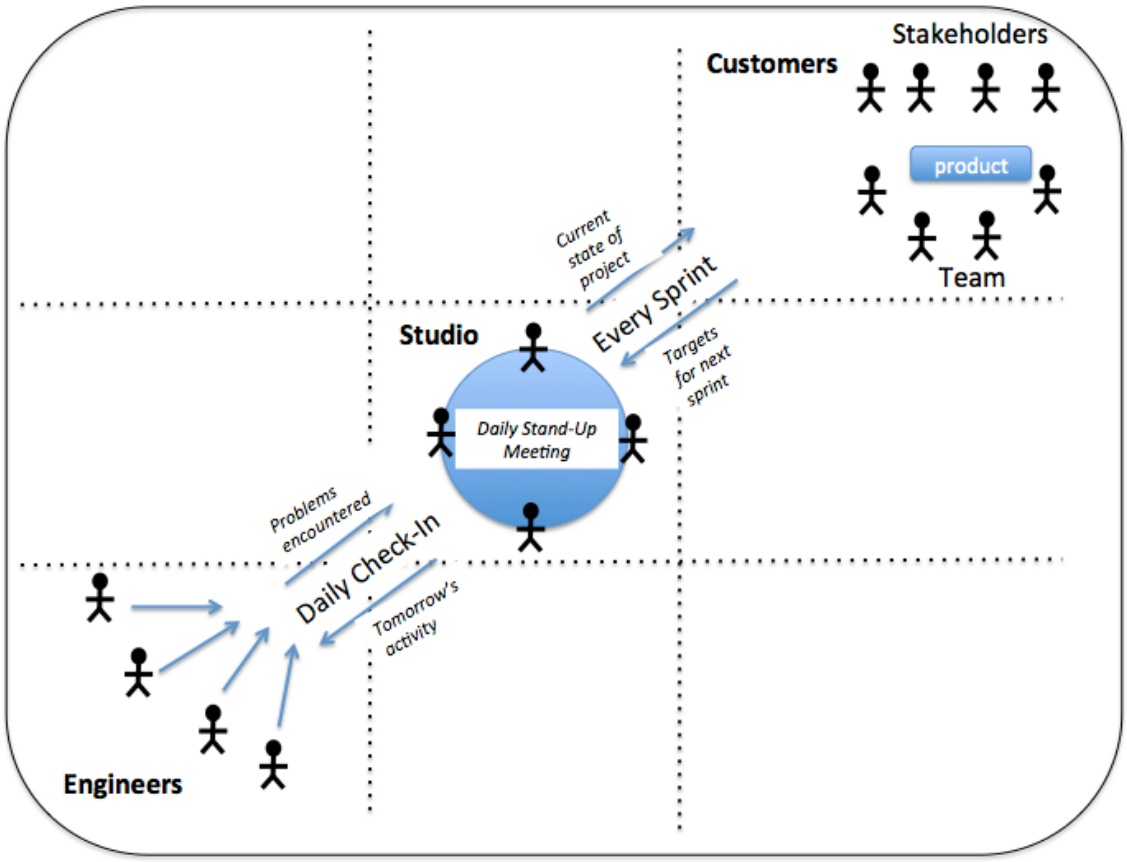

Figure 4: Agile Development as Dynamic Design

Viewed from a dynamic perspective, agile offers two potential benefits over waterfall. First, in waterfall development, the frequency of collaborative episodes is usually too low, both among the team and between the team and its customers. A developer working for a week or two without a check-in could waste considerable effort before it's clear that he has made a mistake or gone off course. This creates a situation akin to pre-Toyota manufacturing in which an operator might have produced many hours worth of defective parts before the underlying problem was identified by the quality department. Developers often do not wait this long and informally check in with supervisors or teammates. While seemingly functional, these check-ins can lead to a situation in which the entire team is not working off a common base of information about the state of the project. In this case, the operating mode starts to migrate from the box on the lower left, the "factory" mode, to the one on the lower right, which results in costly and slow iteration (see sidebar). Research suggests that in R&D processes, this mode can be highly

inefficient.[13] Similarly, checking in with more senior leadership in the form of phase-gate reviews means that the entire team could work for months before realizing that they are not meeting management's expectations, thus also creating potential rework.

Second, escalating from factory to studio mode is only useful if doing so creates a meaningful opportunity to identify and fix otherwise hidden problems. The Andon cord (or the yellow button) would be nothing but a distraction if it didn't lead to defect elimination. The agile approach to software development also improves the quality of time spent in the studio. The focus on developing pieces of functionality means that both the team and the customer are never more than a few weeks away from a piece of software that can be used, making it far easier to assess whether or not it meets the customer's need. In contrast, in waterfall, the early phases are characterized by long lists of requirements and features, but there is nothing to actually try or test. A customer looking at such a list would have a hard time envisioning how that list might look when actually developed into software. It's not surprising that waterfall methods often lead to projects in which major defects and other shortfalls are discovered very late in the development cycle and require costly rework.[14]

Creating Agility Through Dynamic Design: Triggers and Checks

Both the Toyota production system and agile-based software methods are thus examples of what we call good *Dynamic Work Design*. In contrast to traditional, static approaches, Dynamic Work Design recognizes the inevitability of change and builds in mechanisms to respond to that uncertainty. The Andon cord works by allowing operators the opportunity to leave highly specified work immediately when problems occur. Agile-based software methods constitute a more effective dynamic design by both moving the project more frequently between factory and

[13] Perlow, "The Time Famine."
[14] See Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time*, and Kamensky, "Digging Out of the Digital Stone Age."

studio modes and by increasing the effectiveness of those studio sessions for finding problems and misalignments. Of course, both the Andon cord and the agile approach are specific to the nature of the work being done. Once managers recognize the necessity of moving between more individual and more collaborative modes of work, they can build on three ideas to create shifting mechanisms that are well matched to the work of their organization.

First, stripping away all of the hype, the agility of any work process—meaning its ability to both adjust the work due to changing external conditions and resolve defects—boils down to the frequency and effectiveness with which the output is assessed. In both traditional, pre-Toyota manufacturing and waterfall development, the assessments are infrequent and not particularly effective. Consequently, both approaches tend to be slow to adjust to changes in the external environment and quality will only be achieved through slow and costly rework cycles. In contrast, when assessments are frequent and effective, the process will be highly adaptable and quality will improve rapidly. The fundamental recipe for improved agility is:

smaller units of work, more frequently checked.

Second, leaders wishing to create more agile processes can follow a simple two-step approach. First, identify the *help chain*, which is the sequence of people who support those doing the work. In manufacturing and assembly, this help chain starts with the operator and extends from foremen to supervisors all the way up to the plant manager. In software, the help chain often begins with the engineer and moves through the team leader to more senior managers, ultimately ending with the customer. It is critical, in our experience, that you identify the chains of individuals who do and support the work, not their roles, departments, or functions. Work is done and supported by people, not by boxes in an organizational chart or process diagram, and increasing agility requires knowing whom to call when there is a problem or feedback is needed.

Third, once you understand the help chain, you have two basic mechanisms for activating it: triggers and checks. A *trigger* is a "hard-wired" test that reveals defects or misalignment and then moves the work to a more collaborative mode. In our opening example, the operator's inability to complete the assembly on time triggered intervention by the supervisor. Similarly, online retailers use increasingly sophisticated triggers to indicate a misalignment between their product offerings and customer desires. A *check* works in the opposite sequence; the work is first moved to a more collaborative environment and then assessed. In agile software development, this shift happens daily in stand-up meetings where the team quickly assesses the current state of the project. Completing a sprint creates a second opportunity to check in with the customer.

Note that both pre-Toyota manufacturing and waterfall software development contain triggers and checks, respectively. Problems, however, emerge when those mechanisms are either too slow and infrequent or do not catalyze meaningful collaboration with those who actually do the work. In old-school manufacturing, defects would only be discovered by the quality department long after the problem had emerged and thus would create costly rework or warranty claims. Worse, once a defect was discovered, it would be analyzed by engineers without the input of the operators doing the work. Similarly, in waterfall development the checks are often both too infrequent and too superficial to reveal the problems that eventually result in major delays and cost overruns.

In the next section, we demonstrate our approach in action by describing a recent intervention using this framework, which led to significant improvements in both efficiency and adaptability.

4.0 Improving supply chain performance at RefineCo

RefineCo owns several oil refineries and distribution terminals in the United States. In 2010 and 2011, in response to changing market conditions, RefineCo went through a major reorganization that included the sale of several assets. The scrutiny that came with the reorganization revealed a procurement organization that was uncompetitive by almost any benchmark. They paid more for similar parts and services than their competitors, and the procurement group's overhead costs were higher than the industry average. Even more troubling, the refinery was often unable to reliably execute the most basic requirements. When critical parts were not delivered to a particular refinery, it often turned out that the particular location was on credit hold due to the inability to pay the supplier in a timely fashion. Every participant in the system, from senior management down to the shipping and receiving clerks, was frustrated.

In May 2012, the existing director of procurement retired, and a new manager was promoted into the job. A brief tour of the operation revealed a formidable challenge. Although it needed to be improved on essentially every dimension to remain competitive, the procurement system faced no shortage of corporate-level "initiatives" nominally designed to remedy its problems. In addition, the system seemed highly "person" dependent, typically relying on specific individuals who had learned to get things done by working "around" the formal system. Finally, the function's aging demographics suggested that a good fraction of those people would retire in the next three to five years. To meet these challenges, the new director began working with the authors, and it was through this intervention that the initial version of our dynamic contingency model was developed. Interventions were ultimately made at all of the refining sites, but we focus on the first of these, since it contains the two basic insights that were then replicated in different guises at other locations.

The procurement system at each of RefineCo's sites worked roughly as follows: An area or function in the refinery often needed an item like a gasket or a service like hazardous waste

disposal, which could only be performed by an outside vendor. To procure the item or service, a representative from the relevant department would enter the requirements into the electronic procurement system, which would then appear as a request to the central procurement function. The staff in the procurement office would then review the request, identify the appropriate supplier of the good or service, and issue a purchase order. That order would go to the supplier, who would then ship the appropriate product or schedule the delivery of the requested service. When the product arrived at the refinery or the service was completed, a packing slip or service order verification slip would be generated, which would also be entered into the electronic procurement system. Finally, once the item was shipped or the service was delivered, the supplier would generate a payment invoice that was also entered into the system. The electronic system would then perform a three-way match to verify that everything was done correctly: The purchase order should match the verification receipt, which, in turn, should match the invoice. If there was not a three-way match, the invoice would be "kicked out" of the system and the supplier would not get paid until the discrepancy was resolved.

The job of resolving those discrepancies fell to the staff in the refinery's purchasing office. Unfortunately, the products and services procured frequently failed the three-way match, leading to both an overburdened purchasing department and very frustrated suppliers. Though the refinery was part of a large and successful company, it was frequently on "credit hold" with its suppliers for failure to pay invoices on time, making it difficult for the staff to do their jobs and run the plant safely. The dedicated procurement staff worked 10-plus hours per day and had hired temporary workers to help manage the backlog, but they were still falling behind.

The intervention started with interviewing the procurement team. Most of the members complained bitterly about being "overworked" and how "screwed up the system was." Nobody saw any opportunity for improvement beyond adding what appeared to be much needed staff. The procurement system was thought to be the epitome of a process that lacked agility, a

bureaucratic behemoth that couldn't accommodate the slightest departure from normal procedure. The critical moment in the intervention came when one of the longtime team members was asked to compare a good purchase order and a difficult one. A good purchase request contained "all the information I need" and could be turned into an official purchase order in "five to ten minutes." A difficult one, however, missed key pieces of information and might require one to two hours to process as the purchasing staff traded emails with both the requesting unit and the supplier to try to figure out what was needed. Despite this effort, difficult purchase orders were usually the ones that failed the three-way matching process and got "kicked out" of the system. Further investigation revealed that the purchase order system was completely gridlocked with the "kicked out" orders and the team spent much of their time trying to sort them out and clear the backlog. The system had descended into the classic "expediting" or "firefighting" trap: There were so many purchase orders in process, that the turnaround time for any given one was very long. But, long turnaround times created unhappy customers and suppliers who constantly called to complain and figure out where their particular order or payment was. Consequently, the procurement team was constantly reprioritizing its work and reacting to whichever customer or supplier was most unhappy. The situation was only getting worse.

The basic elements of the intervention are shown in Figure 5.

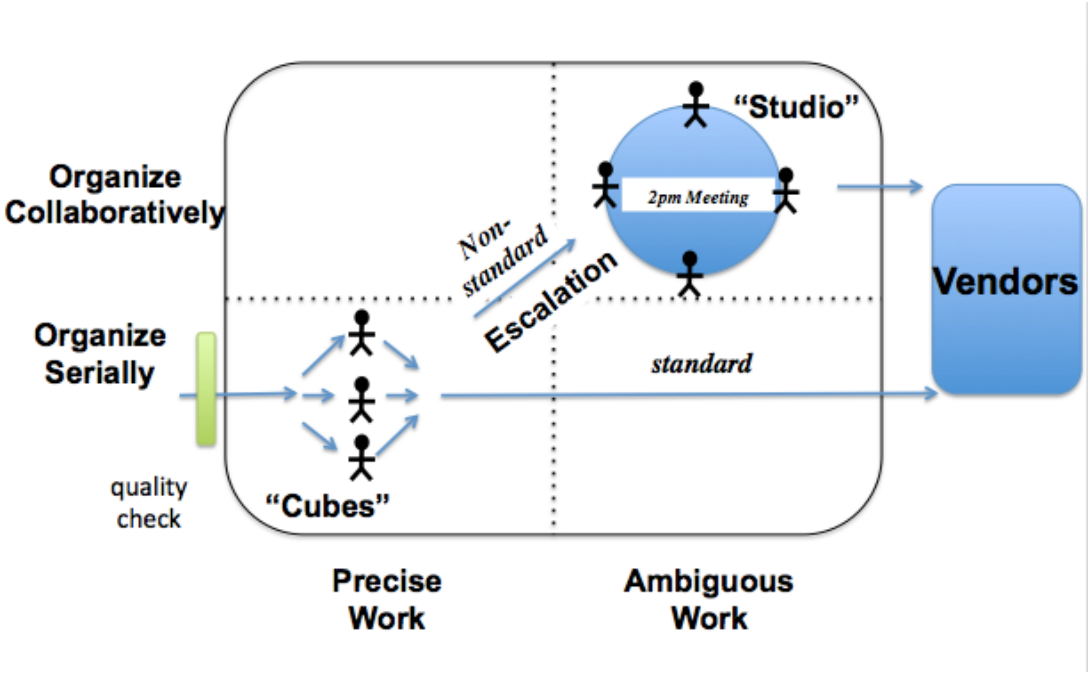

Figure 5: Improving Purchase Order Processing

The first insight (which informed our model) came in recognizing that the procurement team was engaged in two very different types of work that corresponded to what we now call "factory" and "studio." When the requested item was standard and all the needed information was provided, a single person could easily process the request without collaboration; and, once the purchase order was entered, it would easily flow through the system, just like an item on the assembly line. However, the factory wasn't well designed. Standard requests flowed easily through the system *if* the request came with the correct information. If it did not, then it could require several rounds of iteration, usually via email exchange, to issue the purchase order. Making the factory portion work better was straightforward. Factories thrive on standardization, and their efficiency suffers when non-routine elements emerge. So, the purchasing function created a simple checklist that described a good purchase request. The idea was to ensure that standard orders would always arrive at the factory with the correct information. To give the various departments an incentive to use the checklist, the purchasing function promised that any request received by 7am *with the proper information* would result in a purchase order being issued by 2pm that day.

At the time of the intervention, a one-day turnaround was unheard of because every order simply went into the "to do" pile, whether it was for a standard item or a highly complex one for which there was no precedent. The purchasing department also created a simple trigger: Purchase orders that were missing items on the checklist would be immediately returned to the requesting unit.

The second part of the intervention came in recognizing that not every request could be supported in factory mode. Refineries are complex places. The slate of products being produced changes regularly due to both supply and demand conditions, and the technology used in production units is often being modified. There is not a precedent for every purchase request, nor is all the needed information always known when a need is identified. Trying to handle these requests via the factory mode created many of the pathologies that the group was experiencing. In the existing system, neither the requesters nor the purchasing staff distinguished between a standard request and a novel one. Thus, when a novel one showed up, the agent would do her best to process it, typically requiring multiple emails with the requester to nail down the relevant information. This process alone could take several days and delay the receipt of the item or service. In many cases, when the agents couldn't get the information they needed, they would make their best guess and then submit an incomplete or incorrect purchase order. This, too, created additional iteration, as the supplier, unsure of what was being requested, would call or email the agent. The purchasing process was thus living in the lower right-hand box of our matrix, attempting to accomplish ambiguous work in a serial fashion and thereby creating slow and expensive iteration.

Creating an effective studio to handle the complex purchase orders had two elements. First, the team created a clear trigger: If a request was non-standard, meaning it could not be processed quickly, then it was moved into a separate pile and not dealt with immediately. Second, each day at 2pm, the team, having finished all the standard requests, would work together to process the

more complex cases. By working collaboratively (in studio mode), they were able to resolve many of the more complex cases without additional intervention—somebody on the team might have seen a similar order before. Also, having a face-to-face meeting was far more efficient than the endless chain of email that it replaced. And, if additional information was needed, the team could schedule a phone call rather than send an email, again reducing the number of expensive iterations.

The results of these two changes were significant. Creating a factory for the standard orders allowed the team to hit its "in by 7 out by 2" promise almost immediately, generating an immense amount of goodwill with the requesters. Spending the afternoon in studio mode also sped the processing of the complex orders. The two changes created enough space that the team was able to use studio time to not only process the more complex requests, but also work through the backlog of orders that had previously been "kicked out" of the system. In the end, due to the efficiency improvements, the procurement team reduced its staff by two full equivalent staff members, while providing far faster and more reliable service. These basic insights were then moved to the other US sites and, as of this writing, RefineCo now pays over 90 percent of its invoices on time, resulting in a far happier collection of suppliers.

5.0 Are you looking for best principles?

Managers, consultants, and business school professors have become obsessed with the search for best practices—those activities that appear to separate leading organizations from the rest of the pack. The idea behind this search is simply that once identified, best practices can be adopted by other organizations, which will then experience similar gains in performance. While there is certainly *some* truth to this idea, the supporting evidence is decidedly mixed. Organizations frequently struggle to implement new tools and practices and rarely experience similar gains in performance. In many industries, the performance gap between the top and middle performers remains stubbornly difficult to close. A key reason for these failures is simply that organizations

are complex configurations of people and technology, and a set of tools or practices that works well in one context, might not be equally effective in a major competitor, even if it is located just down the street.

At the same time, though organizations are often very different, they share common elements by virtue of being inhabited by human beings who come with a common set of strengths and limitations. Best practices are "best" when they manifest an underlying behavior principle in a way that is well matched to the organization that uses them. Toyota's famed Andon cord and the localized problem solving it catalyzes work, because this system capitalizes on the efficiency that comes from individual repetition and the innovation that comes with collaborative problem solving. Conversely, agile development methods work by channeling the creativity of software engineers through frequent team meetings and customer interactions. More generally, organizations become more adaptable when they find defects and misalignments sooner.
A dynamic approach to contingency, supported by triggers and checks, can open the path to creating practices that support increased agility in the work of *your* organization.

SIDE BAR

Two Failure Modes

What happens when organizations don't do a good job of cycling between factory and studio? We observed two related failure modes, *ineffective iteration* and *wasted attention*.

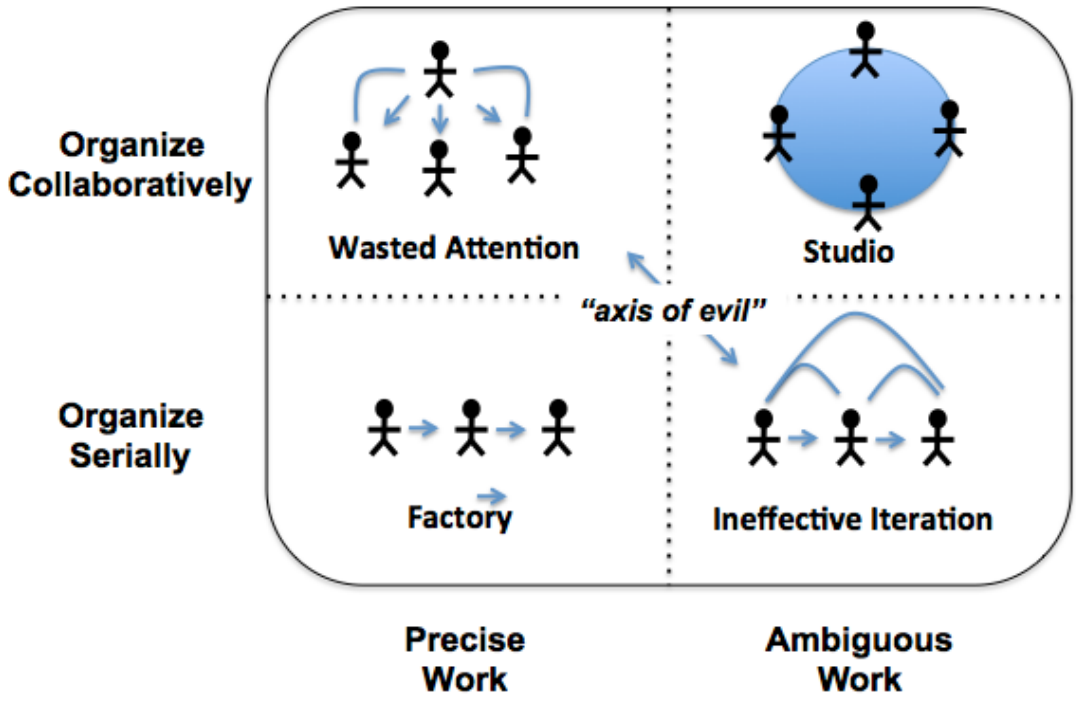

Figure 6: Two Failure Modes

Ineffective Iteration. Consider first what happens when elements of the work in question are highly ambiguous, but are nonetheless organized serially (captured in the box in the lower right-hand corner). This approach tends to create slow and costly iteration relative to a more collaborative design. The lack of speed comes because the ambiguity must travel among participants to be resolved, thus requiring multiple rounds, each of which takes time. Worse,

when knowledge work is designed serially, many of these interactions take place through email or text messaging—"I'll email you the spreadsheet after I am done with it." Research suggests both that such communications modes are less effective for ambiguity reduction than face-to-face communication *and* that those sending such messages are unaware of those limits[15]. As everybody knows intuitively, trying to resolve ambiguity via email or text messaging tends to create more misunderstandings and often necessitates multiple iterations.

Wasted Attention. On the flip side, organizing precise work in a collaborative fashion also creates inefficiency. If the work is precisely defined, then it doesn't benefit from multiple eyes throughout the process, despite substantially multiplying the cost. Worse, too much collaboration may prevent the efficiencies that come with the learning curve that emerges when people repeat the same task.[16] Consider standard meeting practice. Often, meetings are one-way information transfer, so everybody is on "the same page," but very little ambiguity or uncertainty is actually confronted. Consequently, such meetings are tedious, and it can be difficult to stay engaged. Worse, attendees who are not actively participating (because there is no problem that needs to be solved or decision that needs to be made) often engage in a variety of unsanctioned behaviors (now greatly enabled by the ubiquity of smart phones) that undermine the meeting's purpose.

"The Axis of Evil"

Whereas functional work processes move between the factory and studio modes (based on triggers and checks), our research suggests that absent careful design attention, processes can devolve to the point where they move between the failure modes described above, oscillating between wasted attention and ineffective iteration, a dynamic that we call "The Axis of Evil."

[15] Kruger and Epley, "When What You Type Isn't What They Read."
[16] Argote and Epple, "Learning Curves in Manufacturing."

Getting stuck on the axis of evil typically starts with time pressure—a project is behind schedule or a more repetitive process is not delivering on its targets. When people feel they are behind, they don't want to take the time to escalate into problem solving (or studio mode), preferring to stay in the factory box on the lower left and "just get the work done." The consequence of this decision is to leave one or more problems unresolved, whether it be an element of a product design that doesn't work or a defect in a manufactured product. Eventually, these problems will be discovered, usually by an activity downstream from the one that generated it. And, if this problem is not then escalated (again due to time pressure), but instead sent back for rework, then the system has effectively moved from the box on the lower left to the box on the lower right and is now in "ineffective iteration" mode.

The consequence of ineffective iteration is that the process becomes increasingly inefficient and incapable of meeting its targets. Senior leaders are, of course, unlikely to stand idly by and will eventually intervene. Unfortunately, the typical intervention is often to scrutinize the offending process in more detail, usually in the form of more frequent and more detailed review meetings. As a manager we once interviewed said, "I knew my project was in trouble when I was required to give hourly updates." But, the form of those reviews makes all the difference. If they are well designed and focus on resolving the key problems that are creating the iteration, then they can move the system back to a better cycling between factory and studio modes. These interventions, however, are the exception rather than the rule.

Most processes have not been designed with escalation mechanisms in mind. So, when senior managers want to intervene and scrutinize a project, they don't know where to look and want to review *everything*. The result of such scrutiny is long review meetings, the majority of which focus on elements of the process that are just fine, thereby trapping the process in the upper left-hand box, "wasted attention." Worse, long review meetings and the preparation that they require steal time and resources from actual work, thus intensifying the time pressure that prevented

proper escalation in the first place. Without careful attention to the mechanisms that move a process between the individual and collaborative modes, processes can increasingly cycle between ineffective iteration and wasted attention, basically moving between frantically trying to solve (or at least hide) the latest problem before the next review, and endless, soul-destroying review meetings that never get to solving the problems that would really make a difference.

Argote, Linda, and Dennis Epple. "Learning Curves in Manufacturing." *Science;* Washington 247, no. 4945 (February 23, 1990). http://search.proquest.com/docview/213555502/abstract/58B71843BC6F455DPQ/1.

Brown, Tim. *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. New York: HarperBusiness, 2009.

Burns, Tom, and G. M. Stalker. *The Management of Innovation*. Rev. ed. Oxford ; New York: Oxford University Press, 1994.

Epley, Nicholas, and Justin Kruger. "When What You Type Isn't What They Read: The Perseverance of Stereotypes and Expectancies over e-Mail." *Journal of Experimental Social Psychology* 41, no. 4 (July 2005): 414–22. doi:10.1016/j.jesp.2004.08.005.

Glaiel, Firas S., Allen Moulton, and Stuart E. Madnick. "Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods." Working Paper. Massachusetts Institute of Technology. Engineering Systems Division, October 2014. http://dspace.mit.edu/handle/1721.1/103024.

Goldstein, Steven D. "Apply Agile Development to Every Aspect of Business." *Mint*, December 4, 2016. http://www.livemint.com/Companies/XvXxgQFUUJWueVegofpknM/Apply-agile-development-to-every-aspect-of-business-Steven.html.

Kamensky, John. "Digging Out of the Digital Stone Age." *Government Executive*, March 9, 2017, 1–1.

Lawrence, Paul R., and Jay William Lorsch. *Organization and Environment: Managing Differentiation and Integration*. Rev. ed. Harvard Business School Classics 1. Boston: Harvard Business School Press, 1986.

Leonard, Devin, and Rick Clough. "How GE Exorcised the Ghost of Jack Welch to Become a 124-Year-Old Startup." *Bloomberg.Com*, March 17, 2016. https://www.bloomberg.com/news/articles/2016-03-17/how-ge-exorcised-the-ghost-of-jack-welch-to-become-a-124-year-old-startup.

Leveson, Nancy. "A Systems Approach to Risk Management through Leading Safety Indicators." *Reliability Engineering & System Safety* 136 (April 2015): 17–34. doi:10.1016/j.ress.2014.10.008.

Liker, Jeffrey, Michael Hoseus, and Center for Quality People & Organization. *Toyota Culture: The Heart and Soul of the Toyota Way*. New York: McGraw-Hill Education, 2008.

Moreira, Mario E. *Being Agile: Your Roadmap to Successful Adoption of Agile*. First edition. New York: Apress, 2013.

Perlow, Leslie A. “The Time Famine: Toward a Sociology of Work Time.” *Administrative Science Quarterly*, no. 1 (1999): 57.

Repenning, Nelson R., and John D. Sterman. “Capability Traps and Self-Confirming Attribution Errors in the Dynamics of Process Improvement.” *Administrative Science Quarterly* 47, no. 2 (June 2002): 265–95.

Stettina, Christoph Johann, and Jeannette Hörz. “Agile Portfolio Management: An Empirical Perspective on the Practice in Use.” *International Journal of Project Management* 33, no. 1 (January 2015): 140–52. doi:10.1016/j.ijproman.2014.03.008.

Sutherland, Jeff. *Scrum: The Art of Doing Twice the Work in Half the Time*, 2014.

Ulrich, Karl T., and Steven D. Eppinger. *Product Design and Development*. Sixth edition. New York, NY: McGraw-Hill Education, 2016.

West, Dave, and Tom Grant. “Agile Development: Mainstream Adoption Has Changed Agility.” Forrester, January 20, 2010. https://www-forrester-com.libproxy.mit.edu:9443/report/Agile+Development+Mainstream+Adoption+Has+Changed+Agility/-/E-RES56100.