

Managerial Implications and Comparative Effects of SAFe Scaled Agile Methods in Government Software Acquisition

Allen Moulton

MIT Sloan School of Management

Sean Ricks¹

The MITRE Corporation

John James¹

The MITRE Corporation

Greg Love¹

The MITRE Corporation

Aleksandra Markina-Khusid¹

The MITRE Corporation

Greg Howard¹

The MITRE Corporation

Paula Mahoney¹

The MITRE Corporation

Stuart E. Madnick

MIT Sloan School of Management

Abstract

As Lean-Agile software development methods have become more prevalent at the practitioner level, an important question has arisen about how these small team methods can scale to meet the needs of larger organizations from the department to the enterprise level. Dean Leffingwell's *Scaled Agile Framework for LEAN Software Development* (SAFe) incorporates a number of Lean-Agile best practices and has become a leading proposition for how to achieve these ends by scaling the operational model for software creation and delivery at scale. One of the questions that managers, and particularly government managers, raise is how Lean-Agile practices such as SAFe can fit within their organizations and meet the control regulations and requirements with which they must comply. To help these managers visualize the application of Lean-Agile practices using SAFe and how it might work for them, we have used System Dynamics modeling to investigate the operational characteristics of SAFe and build a prototype model that can be used as a learning tool and as the basis for future extension into decision support simulations to evaluate alternative organizational transformations based on SAFe. The key objective is qualitative understanding of the implications of possible scenarios rather than precise numeric metrics. The System Dynamics model-based investigation also allows SAFe to be compared structurally to other alternative approaches.

1. Introduction

As software has become more and more important to every aspect of modern life, the delay inherent in traditional software practice has become a critical liability. Managers in corporate and government sectors are seeking better methods to deliver quality software faster to dramatically cut the economic cost of delay. In defense and intelligence, the cost of delay can extend beyond economics to human cost when lives are placed at risk.

Concern about excessive delay in large scale government and defense software acquisition has been expressed repeatedly for over thirty years. A 2009 Defense Science Board report found that the average time from concept to initial operating capability was 91 months, that is, over seven years. In 1982, a defense study recommended an Agile-like process to “buy a little, test a little” to minimize overall software development exposure and risk (AFCEA, 1982). Thirty years later,

¹ The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

a 2012 Brookings study found a “growing consensus on the need for agile approaches” in defense IT system acquisition (Duvall, 2012)

Agile methods have come to be seen as a likely means to break the bottlenecks in traditional processes. Agile grew from small teams building SmartPhone apps and other small software products to larger scale projects and increasingly to the software across the entire enterprise. SAFe, the Scaled Agile Framework (Leffingwell, 2017) has emerged as a leading contender for scalable practices that can achieve Agile benefits in organizations large and small.

A 2010 study by SEI of Agile in defense software acquisition was pessimistic, finding that while it was theoretically possible to use Agile in DoD, there were no regular practices established and program managers had difficulty fitting Agile into the DoD control processes (Lapham, et al., 2010). A follow up study in 2016 was decidedly more hopeful, finding that regulations has been revised and that “it is increasingly clear that Agile can produce a better product developed within cost and schedule parameters when both the government and the development side understand how to effectively employ Agile principles and practices” (Miller & Ward, 2016, p. v).

While the anticipated benefits are large, adopting radical changes in how software is developed and delivered brings with it risks that the change may not work out successfully in a given organizational setting (*viz.*, Elbanna & Sarker, 2015). A manager faced with this kind of risk wants to be able to test drive the vision of the future before committing to large transformations in the way the organization works. Managers also frequently only have control of the part of the organization over which they are responsible and can at best make recommendations for changes to the larger context around them.

In order to make SAFe or other Agile methods work in a government context, and in many private sector enterprises, a manager may need a hybrid methodology. For example, a DoD program manager must live within the large-scale processes such as the Planning-Programming-Budgeting-Execution (PPBE) system that formalizes planning for future requirements, programming to assign requirements to future years, funding for the next year, and execution in the current year (Army Force Management School, 2011). This structure, and others that constrain manager flexibility, are decidedly un-Agile.

System Dynamics offers the potential to give managers a way to visualize the outcome of new methods before making a commitment. In this paper we describe a structural model that managers can use to walk through new ways of doing business. In follow on work, we will operationalize the model to allow managers to simulate the behavior of alternative choices in how to introduce Agile and integrate it with other elements and constraints within their organizations. We will also extend the model from the specifics of SAFe to hybrid methods.

Prior Research

The seminal work on applying System Dynamics to Software Development was the Abdel-Hamid and Madnick (1991) model of large Waterfall software development projects. In that model, the scope of the deliverable software was fixed as was the intended delivery date. The model addressed variation in cost and schedule lateness depending on a number of policy

variables concerning how staffing and other management decisions were made. Madachy (2008) includes a variety of System Dynamics models of different aspects of software development.

In an early Agile study, Chichakly (2007) compared Agile to Waterfall methods with an emphasis on the effects of short Agile cycle times and automated testing on project performance under stable and varying requirements patterns, finding Agile superior except for quality concerns. Tignor found that the Rework Cycle is applicable to Agile (2009) and that single and double loop learning effects are important to Agile success (2010). Cao (2010) looked at the commercial sector Extreme Programming (XP) and Scrum methods, and the interaction of customer involvement, pair programming, and refactoring. Marjaie and Rathod (2011) applied grounded theory to model communications effects in Agile. Kong et al. (2011) used a feedback loop to examine the effects of schedule pressure in Agile software maintenance processes. Cocco et al. (2011) used a model with a two-stage backlog with a rework cycle to explore the effects of Kanban and Scrum compared to Waterfall. Glaiel et al. (2013) developed the concept of an Agile Genome of significant dimensions of Agile development and used that in a System Dynamics model with modular sub-models for different genes.

System Dynamics has also been applied to related problems of executing projects through contractors (McKenna, 2005), agency theory effects in government acquisition of large scale ERP systems (Moulton, et al., 2015), the unexpected effects of funding fluctuations on software delays (Trammell, et al., 2016), and many other related concerns.

2. Scaled Agile Framework (SAFe)

SAFe, the Scaled Agile Framework (Leffingwell, 2017) is recognized as a best practice for Lean-Agile implementation at enterprise scale. The Version One State of Agile Survey (Version One, 2016) found that “SAFe saw the largest increase from 19% in 2014 to 27% in 2015 making it the second most prevalent Agile scaling method cited by respondents.”

In a study of the potential for Agile software development in DoD, Miller and Ward concluded that “SAFe is particularly worth a closer look because it directly addresses some of the primary barriers to adopting Agile within a large enterprise such as the DoD as well as being a framework that program offices are likely to see proposed by defense contractors” (2016, p. 10).

SAFe combines a number of Lean-Agile best practices into a holistic framework for the Lean-Agile enterprise. SAFe includes all aspects of a Lean-Agile enterprise (e.g., software development, requirements, planning and scheduling, budgeting, modeling, contracting, quality, test, etc.). SAFe is both scalable and extensible in that not all levels and aspects need to be adopted at once.

The SAFe 4.0 White Paper is an excellent overview of the framework describing SAFe as

a freely revealed knowledge base of proven, integrated patterns for enterprise-scale Lean-Agile development. It is scalable and modular, allowing each organization to apply it in a way that provides better business outcomes and happier, more engaged employees. SAFe synchronizes alignment, collaboration, and delivery for large numbers of Agile teams. It supports both software and systems development, from the modest scale of well

under 100 practitioners to the largest software solutions and complex cyber-physical systems, systems that require thousands of people to create and maintain. SAFe was developed in the field, based on helping customers solve their most challenging scaling problems. It leverages three primary bodies of knowledge: Agile development, Lean product development, and systems thinking. (Scaled Agile Framework, 2016b, p.1)

There are four SAFe core values: alignment, built-in quality, transparency, and program execution.

- Alignment refers to “taking a systems view” over seeking to achieve local excellence at the expense of the global objective.
- Built-in quality refers to achieving a fast, continuous flow with a minimum of delays due to rework, high value delivery velocity, and the highest levels of customer satisfaction.
- Transparency ensures that all stakeholders are on the same page and serves to enable trust.
- Program execution refers to routinely and predictably delivering value.

The SAFe Lean-Agile mindset draws on the work of lean manufacturing (Roos et al., 1991; Anderson, 2010) depicted as the “House of Lean” (Leffingwell, 2017, p.28). The SAFe Lean-Agile Goal is “delivering maximum customer value in the sustainably shortest lead time.” The goal is supported by four Pillars: 1) respect for people and culture, 2) flow, 3) innovation, and 4) relentless improvement. The Foundation on which the Lean-Agile house is built consists of managerial leadership. SAFe also draws on the four values of the Agile Manifesto (Beck et al., 2017):

- **Individuals and interactions** vs. processes and tools
- **Working software** vs. comprehensive documentation
- **Customer collaboration** vs. contract negotiation
- **Responding to change** vs. following a plan

The distinctions between the phrases on the left and those on the right illustrates how Agile is different from traditional software acquisition.

In traditional Waterfall software development project management, design, elaboration, and planning of the master schedule for the project are all completed and approved before software development work starts. Given a fully elaborated detailed plan, people are brought into the project and assigned to tasks in the schedule. This usually results in a varying level of staffing and reassignment of staff from one task to another as the project proceeds. If a project falls behind schedule, managers may add additional staff or negotiate to revise the plan to cut scope or stretch scheduled delivery dates. In short, the work in the plan is the independent variable, while staffing is a dependent variable that varies over time to meet the needs of the plan.

Agile uses a different paradigm. As expressed in the SAFe framework: Team size and the Program Increment and Sprint completion dates cadence are set independently. The dependent variable is flow – the velocity at which fixed size teams working in fixed cadence produce working software. Teams determine the scope of work done as the project progresses via a highly structured and coordinated, but decentralized planning process.

The planning process in SAFe elaborates the Lean-Agile concepts with nine principles (Leffingwell, 2017, p.35):

- 1) Take an economic view to delivering value early and often considering the trade-offs between risk, cost of delay, operational and development costs, with the implementation of decentralized decision-making.
- 2) Apply systems thinking to taking the “larger view” of problems and challenges.
- 3) Assume variability, preserve options for maintaining multiple design options to allow for effective mid-course corrections throughout the lifecycle.
- 4) Build incrementally with fast, integrated learning cycles for frequent integration points to provide feedback. This is in contrast to the traditional approach in which test and integration are sequential phases.
- 5) Base milestones on objective evaluation of working systems with integration points serving as objective milestones to evaluate the solution. This is in contrast to the traditional approach of phase gate milestones that produce a single large solution at the end of a series of sequential steps.
- 6) Visualize and limit work in process (WIP), reduce batch sizes, and manage queue lengths for the application of Lean best practices to maintain demand-capacity balance and continuous product flow.
- 7) Apply cadence, synchronize with cross-domain planning for achieving synchronization by aligning the activities of the Value Stream, Program, and Team levels to a common cadence.
- 8) Unlock the intrinsic motivation of knowledge workers acknowledges that while under compensation can be a de-motivator, knowledge workers are motivated by self-actualization rather than monetary compensation such as management by objectives.
- 9) Decentralize decision making because it’s effective to centralize strategic, long range decisions; all other decisions should be decentralized. This serves to shorten decision cycles and enables agility in responding to change.

SAFe provides an all-encompassing model for how to organize and execute software creation and delivery across the whole enterprise. Many organizations may find it difficult to adopt the whole of SAFe at one time and may need to draw on themes from SAFe to use in a hybrid organizational model.

The SAFe “Big Picture” (see Appendix A), is a web-based diagram connecting to an interactive body of knowledge (Scaled Agile Framework, 2016a). There are two variants to the Big Picture: a three-level SAFe is suited for non-complex organizations requiring a small number of Agile teams, and a four-level SAFe suited for large complex organizations involving hundreds or more practitioners. The four SAFe levels are shown in Table 1.

Level	Development Unit	Function	
Portfolio	Epics	Enterprise management	
Value Stream	Capabilities	Funding for solutions	(Optional Level)
Program	Features	Coordination of development	} Agile Release Train (ART)
Team	Stories	Software production (DBT)	

Table 1: SAFe Levels

Starting from the bottom, the Team level is composed of integrated multi-function Agile teams that conduct the define-build-test (DBT) activities that produce functionality. Teams work on writing software defined in Stories, which is an Agile term for the functionality and context of the software to be created. SAFe provides for using several Agile development methods: Scrum, Kanban, and XP. Scaling is provided by organizing multiple Teams into the larger management structures at higher levels.

The Program level provides a Lean-Agile planning and management structure which delivers product Features to the customer. The combination of the Team and Program levels is known as an Agile Release Train (ART). ARTs are limited in size to 50-125 people in order to maintain the necessary social cohesiveness for face-to-face planning and execution. The ART has all the resources to “move ideas from concept to deployment”. Teams within the ART are aligned to a common mission and vision through a single program backlog. The Program Increment (PI) described below provides for synchronization through a common planning and development cadence (nominally 10 weeks).

The optional Value Stream level, a planning and management structure for multiple supporting ARTs, supports the development of large and complex solutions. At the Value Stream level, Solutions deliver Capabilities, which are each a combination of Features. There are three possible Value Stream – ART configurations:

- 1) Multiple smaller Value Stream Solutions provided by a single ART;
- 2) A large Value Stream Solution provided by a single ART; and
- 3) Multiple ARTs coordinated by a Value Stream providing a complex (e.g., systems of systems) Solution.

The three-level variant of SAFe for smaller scale organizations leaves out the Value Stream level between the top Portfolio level and the lower Program and Team levels.

The Portfolio level contains the value streams, people, and processes necessary to provide funding and governance for the products, services, and solutions required to fulfill the overall business strategy (Leffingwell, 2017, p.411). Work is managed in larger constructs called Epics at the Portfolio level of SAFe. The number of Portfolio levels is determined by the size and complexity of the organization. Less complex organizations may only have a single Portfolio level; whereas, large complex organizations may have multiple Portfolio levels.

3. System Dynamics Modeling of SAFe

We began our Scaled Agile investigation by incrementally developing a System Dynamics model starting at the bottom level of the scaling hierarchy, i.e., the Agile Team level where software is produced. Our Agile Team System Dynamics model structure is illustrated in Figure 1 below. In addition to the core structure of the model described here, there are also a number of sub-models for various factors affecting production which are not detailed. These sub-models connect into the main production model through shared variables and are designed to be modularly replaceable.

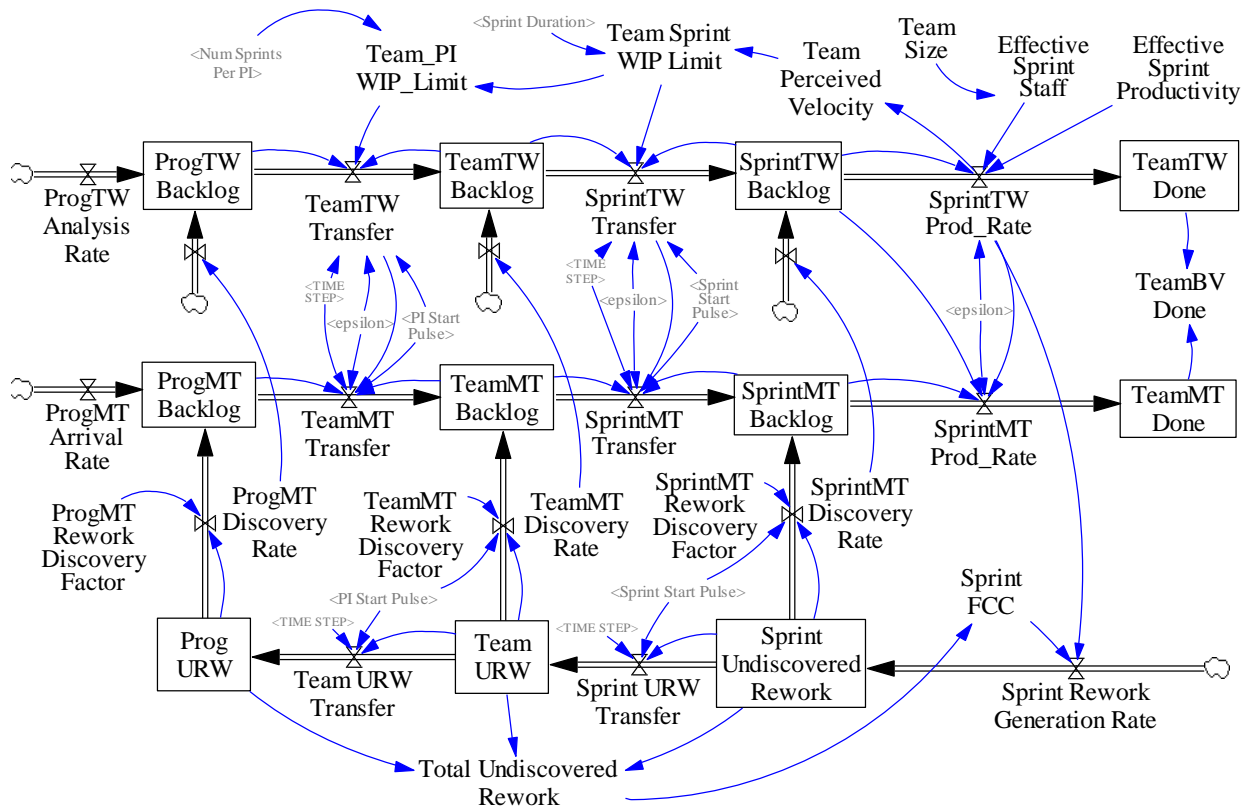


Figure 1. Team Level Model

Agile Teams and Agile Release Train

The Agile Team is generally a multi-function, multi-skilled group of 5-9 individuals who define, build, and test increments of deliverable software in a time boxed interval, which we refer to here as a Sprint. Although the SAFe framework uses the term “increment”, we adopted the more common name “Sprint” since there are several types of “increments” in SAFe and we want to clarify that we are referring to the interval for the Team work increment.

The Agile Team acts as a self-organizing integrated unit performing all the functions of software development. The Defining function involves elaborating the details of a requirement and designing the software that needs to be created for a component or feature. Building involves writing or modifying code that implements a component or feature. Testing involves designing and creating test cases and executing tests to assure that the software built performs as required to meet the needs of the user.

The Team is intended to stay together for the long term. Work is flowed through fixed size Teams. A *Team Size* of 5-9 people is recommended in the SAFe framework. A Product Owner augments the Agile Team to provide guidance on understanding and coordinates prioritizing the work done during each interval. The Product Owner acts as the surrogate for the customer and has the responsibility to sign off on completion of each piece of work done. The Product Owner acts as an important member of the Agile Team, but does not actually produce software.

The amount of parallel work that can be effectively controlled within an ART depends, among other things, on the degree of independence of the software being produced by each Team. The architecture of the product being developed will influence the number of Teams that can work simultaneously without running into deconfliction and coordination problems that must be addressed to proceed at full speed.

Measuring Work: Stories, Story Points, and Work Units

A model needs to have a measure of work flowing through the process. A key concept in most forms of Agile is the Story, which is used at the Team level to define what is needed. A Story is generally written in “user voice” in a statement of the form:

As a <user role> I can <do activity> in order to <produce some value>

By stating Stories in this form, the team is informed about who is going to use the system being developed, what they are going to do with it, and how that will provide a benefit. The Agile Team has the responsibility to determine how to achieve the desired functionality. Acceptance Test Driven Development (ATDD) adds an additional element to the Story to specify the verification criteria for acceptance (Leffingwell, 2017, p.149). ATDD supports automation in the acceptance testing process.

Story size is estimated in Story Points. The usual practice is to use Fibonacci numbers (1, 2, 3, 5, 8, 13) to estimate relative size of the work to accomplish a Story. SAFe continues the sequence with larger numbers 20, 40, 100, etc. Since each Team can decide on its own scale for Story Points, there is considerable controversy about how to compare one Team’s Story Points to another and how to relate Story Points to units of required effort (person-days, etc.). The SAFe framework recommends using Normalized Story Points by finding some small Stories that look like they would be achievable in half a day development and half a day test (i.e., one person-day). Those Stories are then assigned one Story Point and act as a unit measure with all others scaled in relation to them. For common baseline planning purposes, SAFe also suggests using an estimating rule of eight Story Points per two-week interval for each production member of a Team (Leffingwell, 2017, p.139). That works out to an estimating rule of four Story Points per person-week.

To avoid being caught up in differences of opinion on size of a Story Point from one organization or Team to another, we have adopted an independently defined Work Unit (WU) for our model. We define the Work Unit as the nominal production of one person in a week with a scale factor of four. In other words, one person working at full speed could potentially produce four WU per week. It is important to recognize that the WU measures nominal production, not actual production, which will be determined by the operation of the actual process in reality or of the model in a simulation. Our WU work unit measure has been scaled to be roughly equivalent to the unit measure of Story Points as recommended in the SAFe framework. The actual relationship to Story Points in any given organization or program at any given time would vary and would have to be calibrated empirically.

Time-boxes, Program Increments, and Cadence

All parts of the production process at the Team level work on a common cadence. For example, if the basic cadence is a two-week cycle, then all processes work in intervals that are a multiple

of two weeks and all intervals are aligned so that smaller intervals fit inside larger intervals. The variable *Sprint Duration* in our model defines the length of the smallest interval. The SAFe framework documentation recommends a Sprint length of two weeks, although we have seen four-week Sprints in some cases we have examined, and the length may vary depending on what works best for an organization.

Agile uses a time-box principal for managing work. Stories that are accepted into a Sprint are to be completed during the Sprint interval. Incomplete stories should be an exceptional circumstance. If a Story cannot be completed within the Sprint, several things may happen. The Story may be broken into smaller stories, some of which are completed, and some of which remain incomplete at the end of the Sprint. For stories that are incomplete, the Team may decide to hold the Story over to be done in the next Sprint, replan the Story to a different future Sprint, or reconsider whether the Story needs to be done at all. This decision is made at the end of the Sprint as the Team moves forward to the next interval.

The Sprint is the lowest level cycle in SAFe. The next level up is the Program Increment (PI), which is a fundamental work process organizing structure and usually consists of a fixed number of software production Sprints and one Innovation and Planning interval. The SAFe framework recommends a PI with four software production Sprints and one Innovation and Planning interval, for a total cycle of five intervals of equal length per PI. With a two-week Sprint, a PI configured in this way would last ten weeks. The number of production Sprints in a PI can be different from one organization to another, but the SAFe framework recommends that a common PI length be adopted across a given organization. In our model, we capture the length of a PI, measured in terms of Sprint intervals, with the model parameter *Num Sprints Per PI*.

The Innovation and Planning interval serves both as the last step in a Program Increment and the first step of the next PI. The length of the Innovation and Planning interval is captured with the model parameter *InnovPlan Duration*. Teams can do wrap up work on the Program Increment, including completion of left-over work that is undone, final testing, and integration of the work completed into the Solution to be delivered. Also included is an Inspect and Adapt Workshop which includes a PI System Demo to stakeholders, a review of quantitative metrics on how well the Team is meeting expectations, and conducting a retrospective and problem-solving workshop. During the Innovation and Planning interval, Teams can take also time out from relentless production for continuing education and to research and explore innovative ideas that may have future benefit. Finally, the Innovation and Planning interval also includes PI Planning where the Team plans the work that it will take on for the next PI interval with the help of the Product Owner and feedback to the Product Owners.

One of the key features of the SAFe framework is putting the entire development organization on a common work cycle, or cadence, built around the lowest level interval. Having all Teams and Programs on a common cadence provides synchronization and facilitates coordination. We bring cadence into the model with the *Cadence Pulse* variable which is a unit pulse that recurs at the smallest interval width. By assuming that all increments are aligned at an initial point in time, we can create Start Pulses for various levels of increments.

We also introduce a *Cadence Cycle Counter* that accumulates the number of *Cadence Pulses* that have occurred during a model run. This allows us to create other higher order interval pulses. The *PI Start Pulse* signals the start of a Program Increment (PI) which contains a number of Sprint intervals defined by the constant parameter *Num Sprints Per PI*. Since the Innovation and Planning interval is also used as catch up time for incomplete work in the Program Increment, we include it as a normal Sprint for setting the Pulse and then separately control how the time interval is used.

The *Sprint Start Pulse* is also aligned with the *Cadence Pulse*. The first *Sprint Start Pulse* occurs coincident with the *PI Start Pulse* for its Program Increment. Additional *Sprint Start Pulse* events occur at *Sprint Duration* intervals within the Program Increment until *Num Sprints Per PI* are completed. As we saw above, in some Sprint intervals the Team is focused on producing software, while the final intervals in each Program Increment is focused on Innovation and Planning (IP). Although not all Team Sprint intervals operate the same, there is enough commonality to justify a common *Sprint Start Pulse*.

Controlling Work in Process

Another key element of the SAFe framework is Work in Process (WIP) constraints which are used to maximize the flow of work through the production process. WIP constraints are based on Reinertsen's LEAN product development flow (Reinertsen, 2009). Based on Little's Law (Little, 1961), Reinertsen argues that flow of work is maximized and delay is minimized when the quantity of work in the inbound queue is balanced against the capacity of the process doing the work. Reinertsen also argues that the most important metric for evaluating a product production system is economic cost of delay. Business value is maximized by minimizing the economic cost of delay, which combines value to the business of the product being delivered and the timeliness of delivery.

As Nicolette, (2016, p.37) points out, "The math also tells us how to load the queue to avoid stalling the entire process. It turns out the traditional approach of dumping as much as possible onto the development teams and demanding that they work faster, harder, or smarter, only creates delay in delivery. There's a sweet spot where utilization and capacity are balanced, and throughput is maximized." The WIP limits in our model implement this concept and allow experimentation to find the sweet spot.

We set the *Team Sprint WIP Limit* based on the *Team Perceived Velocity*, which is based on the rate at which the Team has been delivering software. The *Team PI WIP Limit* is then based on the *Team Sprint WIP Limit* allowing for one Sprint cycle being non-production. The formulas for deriving *Team Perceived Velocity*, *Team Sprint WIP Limit* and *Team PI WIP Limit* are potential areas for experimentation with the model. As we shall see, the WIP limits are critical factors that control the flow of work through the production process.

Types of Work and the Core Software Production Process

The structure of the Team level of our Scaled Agile model is illustrated by the software production process on the right hand side of Figure 3 along with the related control processes that drive or feed the Team production cycle.

In order to focus on economic cost of delay, we needed to distinguish between work that delivers business value to the customer and other work that needs to be done but does not directly provide value. In investigating Agile with practitioners, we found that there are at least four types of work that flow through an Agile software process:

- BV: Work that delivers business value to the customer;
- EN: Work on Enablers that improve the development process;
- RF: Refactoring work that reorganizes software but does not add functionality; and
- MT: Rework and other maintenance work that fixes problems.

In building our model, we found that we needed to focus on the Total Work (TW) being processed, i.e., the sum of all work categories. We also decided that, for our initial model, we would merge all the non-BV categories into a single category, MT. This would allow us to measure business value created and also to include the Rework that we knew would be there as a distinct category without adding unnecessary detail.

The Team level software production model illustrated in Figure 3, uses two co-flows. The primary flow at the top represents the total work (TW) being done. The second parallel co-flow below includes rework and other non-BV work. At the end of the Sprint cycle on the right, we calculate the BV work completed, *TeamBV Done*, by taking the difference between total work completed, *Team TW Done*, and non-BV MT work completed, *TeamMT Done*. The same calculation of BV could also be done at any point in the process.

The core software production process on the upper right of the diagram uses a standard System Dynamics construct of a material flow from work to be done in *SprintTW Backlog* into work done in *TeamTW Done*. The rate of flow (*SprintTW Prod Rate*) is determined by the product of resources applied (*Effective Sprint Staff*) and productivity (*Effective Sprint Productivity*). Staff is measured in units of FTE-equivalent persons. Productivity is measured in units of WU per person per week. *Effective Sprint Staff* depends on *Team Size* which can be used as an exogenous model policy variable parameter. Actual staff applied to software production itself is reduced to account for the Product Owner who performs an important but non-producing role.

The word “effective” in each variable name indicates that these quantities will vary from nominal FTE staff and nominal productivity rate based on considerations modeled in modular sub-models connected as in the Glaiel Agile Project Dynamics (APD) model structure (Glaiel et al., 2013). For example, effective applied time for staff may follow a conventional assumption of 80% time on task with adjustments for learning curve effects, increased intensity of work under schedule pressure, burn-out effects and other factors. Similarly, productivity may be affected by factors such as development tools and practices that are employed. By separating these components out, we can test alternative representations of important factors while retaining the basic structure of the main SAFe model.

The co-flow shown in the next lower section of the diagram captures the production process for non-business value MT work, which is a proper subset of TW work in the upper flow. This is also a material flow from *SprintMT Backlog* to *TeamMT Done* with a rate of production *SprintMT Prod Rate* that is an allocation of the TW production rate above. The allocation could prioritize MT work over the rest of TW, or it could divide production proportionately to the MT

backlog size compared to the TW backlog, or some other rule for dividing work between the categories. The high level model structure is unchanged as alternative allocation rules are swapped in.

The model also shuts off production when a source backlog drops close to zero (implemented as a value below a minimal quantity, *epsilon*). This prevents backlog level values from falling below zero, which would not be possible in the real system being modeled.

Rework Cycle in Agile

Although SAFe does not explicitly include software rework, prior System Dynamics research has identified the Rework cycle as a critical element in software production performance (e.g., Cooper & Mullen, 1993; Cooper, 1994). Consequently, we incorporated a Rework Cycle in our Scaled Agile model as illustrated on the bottom right of the diagram in Figure 3. We had considerable discussion of how to best include Rework in a time-boxed Agile process model. In a traditional project model, the value to be delivered (scope) is fixed and management questions relate to the schedule (when the product is delivered), cost (how large are the resources expended), and quality (whether the software meets needs and level of defects in the delivered product). A model of traditional development can count up time and cost without worrying about whether the time and cost produced units of value along the way.

In contrast, Agile delivers business value incrementally. We needed to distinguish between time spent on delivering value and time spent on other tasks, such as rework. Rework produces no incremental business value. It just repairs work that was already thought to have been done. By feeding rework into the MT co-flow, we were able to keep true business value production separate so that the cost and schedule of value delivery can be captured distinctly.

The model illustrated in Figure 1 shows *Sprint Undiscovered Rework* being generated by a material flow of work units from a cloud controlled by the *Sprint Rework Generation Rate*. The generation rate is derived from the total work production rate (*SprintTW Prod Rate*) by multiplying by a dimensionless factor representing the fraction of work that is not correct and complete (i.e., $1 - \textit{Sprint FCC}$). The variable *Sprint FCC* is a connecting point into the main model from a modular sub-model that estimates the fraction correct and complete based on considerations such as methods used in development, the level of software complexity, accumulated technical debt, and, as illustrated in the diagram, the invisible quantity of undiscovered rework. Teasing these factors apart empirically may well require considerable effort, but this part of the model provides a standard place to apply that knowledge as it is gained.

Following standard modeling practice, *Sprint Undiscovered Rework* is an invisible quantity that managers know is there and know will require future work to correct problems when it is discovered. Discovery of rework results in a material flow of work units following the arrow up into the *SprintMT Backlog* at a rate of *SprintMT Rework Discovery Rate*, which is implemented as another variable that connects to a modular sub-model that may use factors such as the fraction of Team time devoted to testing and whether continuous integration and automated testing are employed. To keep TW and MT in synch, the same quantity of work units also flows

into *SprintTW Backlog*. As discovered rework is added to the backlogs, it becomes part of the work production cycle in both the TW and MT co-flows.

At the end of the Sprint, any remaining *Sprint Undiscovered Rework* is transferred up to the next level into *Team URW*. Similarly, at the end of the Program Increment, any remaining undiscovered rework is transferred up to *Prog URW*. As rework is discovered at the higher levels it flows down through *ProgMT Backlog* and *TeamMT Backlog* into *SprintMT Backlog* where the corrective work is done. The rate of flow is determined by management policies and controlled by the WIP mechanism described next.

Managing the Flow of Work through Backlogs

The right hand side of our investigatory model (Figure 1) is a continuous process that produces software whenever there is anything in the *SprintTW Backlog*. The left hand side of the diagram works differently. Work is pulled forward into backlogs in chunks at cadence pulse points. The WIP limits discussed above control how much work is placed in each backlog at each pulse.

In the middle of the diagram, the *SprintTW Backlog* is filled via a material flow from the *TeamTW Backlog* at each instant when a *Sprint Start Pulse* occurs. The rate at which work is pulled is shown as *SprintTW Transfer*. This rate is zero except when a *Sprint Start Pulse* occurs. If the *TeamTW Backlog* is empty (less than *epsilon*), nothing happens. Otherwise, a quantity of work is pulled into the Sprint backlog with the constraint that new work added cannot raise the *SprintTW Backlog* above the *Team Sprint WIP Limit*.

At the same pulse time, the *SprintMT Backlog* receives work pulled from the *TeamMT Backlog*, which is a subset of the *TeamTW Backlog*. To maintain the proper subset relationship, a fraction of the *SprintTW Transfer* rate is allocated to the *SprintMT Transfer* rate that controls the flow of work into *TeamMT Backlog*. By maintaining the subset relationship, we also assure that the WIP limit is not violated by MT work pulled into the Sprint backlog.

It is theoretically possible that the *SprintTW Backlog* is already above its WIP limit if a large amount of rework has been discovered and has not yet been worked off. In that case, no new work is pulled into either the TW or MT backlog on this pulse. During the next Sprint, the Team development process performs work and drains the backlog. Since the *Team Sprint WIP Limit* is derived from the perceived velocity of the Team, the process should gradually adapt and adjust so that the *SprintTW Backlog* is empty at the end of each Sprint. At the beginning of the next Sprint cycle, the backlog is refreshed and the process continues.

A similar process happens at the Program Increment level to the left in the diagram. The *TeamTW Backlog* is filled via a material flow from the program backlog in *ProgTW Backlog* at each instant when a *PI Start Pulse* occurs. If the *ProgTW Backlog* is empty (less than *epsilon*), nothing happens. Otherwise, a quantity of work is pulled into *TeamTW Backlog* with the constraint that new work added cannot push the *TeamTW Backlog* above the *Team PI WIP Limit*. The PI WIP limit is set differently from the Sprint WIP limit. At the PI level, the WIP limit is set based on one fewer Sprint cycles than the actual length of the Program Increment (*Num Sprints Per PI*) to allow for an Innovation and Planning interval with no new work at the end.

To see how this works, consider an example from the *SAFe Reference Guide* (Leffingwell, 2017, p.117) of an Agile Team of six with five software production staff and one Product Owner with two-week Sprints and a ten-week Program Increment period. In the example, each developer can produce 8 WU per Sprint giving an estimated Team velocity of 40 WU per Sprint (20 WU per week times two weeks). *Team Sprint WIP Limit* would be 40. At the pulse that begins each production Sprint, 40 WU would be pulled into the *SprintTW Backlog*. At the end of each Sprint, the backlog would be empty (assuming no discovered rework) and the process would continue.

The ten-week Program Increment would be divided into four production Sprint intervals and one Innovation and Planning interval. The *Team PI WIP Limit* would be 160 (4 times 40 per Sprint). At the beginning of the PI, 160 WU would be pulled into the *TeamTW Backlog* to be completed during the PI. If all goes according to plan, each Sprint would use up 40 WU, leaving the PI backlog empty after the four production Sprints complete and the Team would not have any new production work to be done during the Innovation and Planning interval.

This example illustrates how WIP limits are used in the model to control work flow and arrange for an Innovation and Planning interval at the end of each PI with no new work to be done. If things go awry, backlogs will not be fully drained and the Team may have some left over work to complete during the Innovation and Planning interval, which is designed to have slack to absorb variability. There may also be some work left in the *TeamTW Backlog* for the PI, which provides an indicator that the *Team Perceived Velocity* was set too high and needs to be adjusted down. The recalibration of velocity should take place automatically as the Team calculates the work delivered during the PI and recalculates its production velocity.

Although not illustrated on the diagram, a similar process pulls MT work from the program level into the Team level *TeamMT Backlog*. A variant on the illustrated model does a triage on discovered rework, placing less urgent rework into *TeamMT Backlog* and more urgent rework into *SprintMT Backlog*. Although also not shown on the diagram, the *TeamMT Backlog* also receives rework pulled from a program level backlog of rework discovered after the Team has delivered completed software for integration in the ART.

The model is self-adjusting. Left over work is carried forward to the next Sprint and/or PI in the model. The model sees work as a quantity rather than as a collection of objects. In actual practice, the Team would have individual stories in its backlog and would consider what to do with each incomplete Story, whether to break a Story down in to smaller pieces for partial implementation or whether to reconsider the due date of the Story, perhaps pushing it into a later PI. Additional detail, and perhaps agent-based modeling, may be needed to gain greater fidelity on finer grain management of stories.

Managing Work at the Agile Release Train Program Level

To scale beyond the capacity of a single Team, the SAFe Scaled Agile framework adds another organizing construct called the ART, which organizes and integrates the work of multiple Agile Teams to deliver software solutions to the enterprise.

The ART, composed of the SAFe Program and Team levels, is described as a virtual organization of up to twelve Teams that “plans, commits, and executes together” to accomplish a

common mission. In addition to the staff on the Teams, the ART includes other staff that perform roles that cross Teams:

- The Release Train Engineer acts as a leader to coordinate the Teams on common goals;
- Product Management oversees the deliverable content being executed and coordinates with the Product Owner on each Team to actively manage scope and quality as work progresses;
- User Experience designers assure the quality of the product from a user perspective;
- System Architects coordinate how the parts of the solution will fit together and provide guidance to Teams in planning shared components and overall system design structure, which will influence system testing and can impact Team progress when common components need work by multiple Teams;
- The System Team supports infrastructure for development and testing, evaluates compliance with Non-Functional Requirements, performs ART level testing and assists with solution integration and ART System Demos;

Scaled Agile provides for multiple Teams working simultaneously in parallel as part of an ART. To keep coordination overhead within manageable bounds, the SAFe recommends limiting the number of Teams in an ART to no more than eight to twelve, with a total of 50-125 people across the Teams and the ART-level staff (Leffingwell, 2017). One of the key functions of ART-level staff is to assure that the ground is prepared so that Teams always have a right-sized backlog to enable delivery of value at maximum flow rate.

In our model, multiple Teams are represented by replicating the Team model as many times as necessary. Although work at the Program level falls into the categories described above, the discussion here will focus on total work (TW) with the understanding that co-flows similar to the Team level maintain the distinctions between business value work (BV) and other work (MT) with the latter including rework discovered at the Program level as well as below.

The iteration cadence for the Program level is the Program Increment (PI). During PI Planning at the beginning of the PI, each Team pulls work to be done from the common program backlog for the ART in *ProgTW Backlog*. Because of the WIP control mechanism, the total work pulled is the sum of the PI WIP Limits for all the Teams.

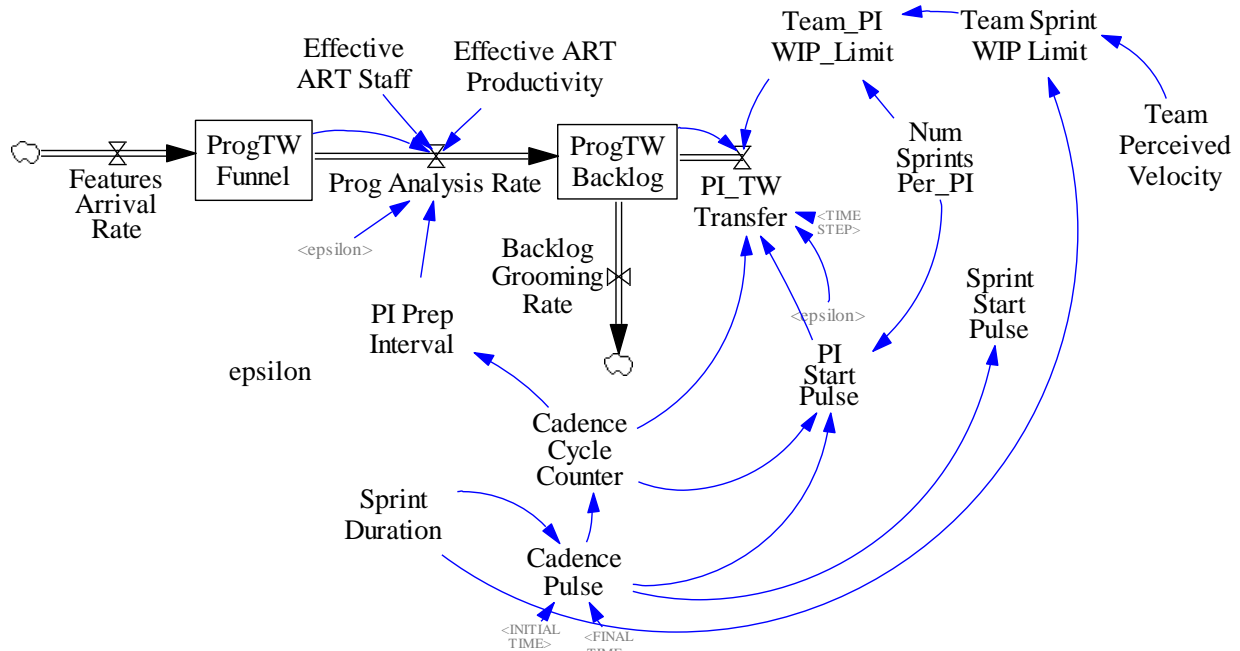


Figure 2. ART Program Level Backlog Refill Process

The Program level staff works through the production cycles of the PI to refill the program backlog by reviewing and analyzing Features in the ART *ProgTW Funnel* and then selecting and decomposing Features into Stories for implementation in a future PI. These Features and Stories are then placed into the ART common program backlog *ProgTW Backlog* for Teams to pull from on the next PI cycle. At the same time, the backlog is groomed to remove Features and Stories that are no longer needed or have lower value. A simplistic version of these activities is illustrated in Figure 2.

The WIP limit on the program backlog is looser than at the Team level, since there must be sufficient work in the program backlog to enable Teams to keep working at full speed and to allow Teams at PI Planning to have a range of options in selecting the work to be done in the next Program Increment. Although the limit is looser, the pull effect is the same. As work is pulled out by the Teams, it must be replaced from the ART backlog in the *ProgTW Funnel*, which in turn creates the need to replenish the ART backlog. This continues the bottom-up pull effect on the whole process.

On the left hand side of the Team model in Figure 1, flow is instantaneous when pulses occur since a quantity of work to be done is already stocked in the sequence of backlogs leading up to the production cycle on the right. In the ART model in Figure 2, however, another analysis process combining staff and productivity must be executed to refill the common program backlog. In order for the whole ART train to keep moving at full speed, management must supply sufficient resources to analyze and decompose Features into Stories to assure that the program backlog is topped up before the next PI begins. The analysis work must also be sufficient to make up for the loss of Stories through backlog grooming.

4. Conclusions and Future Steps

Using System Dynamics modeling, we were able to capture essential structures of the SAFe Scaled Agile methodology, with particular emphasis on the flow of work and value through the software development processes. One of the advantages of System Dynamics is that the model allows visualization of how SAFe would operate when implemented and highlights areas of concern to a manager when considering adoption.

In building the model we had to solve the problem of separating Business Value creation work from Rework in an Agile context when the rate of value delivery and economic cost of delay are key metrics. By creating a dual co-flow model, we were able to maintain this distinction.

Since elements such as production rate and rework generation are based on the Total Work being processed, we had to make one co-flow be Total Work (TW) and the other be other work (MT) while maintaining the subset relationship at all times. The Business Value (BV) work becomes a derived quantity from the difference of the other two types of work.

One of the things we uncovered in our modeling was that WIP limits have the effect of pulling work through processes, Kanban style, rather than being pushed as in traditional processes. This means that the rate of work at the bottom Team level drives the whole process. We found that the model simplified tremendously when we took advantage of WIP limits to control flow. For a manager, this means that setting WIP limits will be critical to the results obtained. As a simulation, the model can be used as a learning tool to understand the implication of SAFe and to test alternative policies for setting WIP limits and parameters that can affect the balance of work flow.

Going forward, we intend to extend the production model to include software delivery and independent testing and verification that may be required in government programs. We will also add more detail for additional SAFe processes and investigate how rework and WIP interact as well as experimenting with plugging in alternative modular sub-models. With the inclusion of the sub-models, we will investigate non-linear effects, such as variations in team size, batch size, and coordination effects within the team and across teamsize an ART.

We also want to address additional questions that have been raised in interviews with managers considering adopting Agile. First, how can a manager exercise control over the complex of teams, ARTs, and other elements making up a SAFe organization? Given that the Agile philosophy is bottom-up self organization of work, how can a manager measure progress to promised deliverables? What can a manager do to influence the SAFe organization? In fact, can a manager exercise control at all? By building models of possible control scenarios meshed to an extended SAFe model, we hope to be able to provide managers with confidence on these questions.

Second, how does SAFe work if the teams are made up of contractors and the Program/ART level is made up of government personnel? What are the agency relationships? How does the SAFe structure achieve the connections between levels needed to align incentives and achieve maximum value delivered?

Third, how can an ART mesh with the kinds of periodic process expectations of bureaucratic management levels above the software organization? In particular, can SAFe facilitate meeting Waterfall-like process deliverables required by higher leadership levels? There is evidence that Agile can work in this environment, but how does it work? A model can help managers answer this type of question.

References

- Abdel-Hamid, T., and Madnick, S. (1991). *Software Project Dynamics: An Integrated Approach* (p. 288). Prentice Hall.
- Anderson, David J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.
- Armed Forces Communications and Electronics Association (AFCEA 1982), *Command & Control (C2) Systems Acquisition Study Final Report*. Washington, DC: Defense Technical Information Center, Sept. 1982.
- Army Force Management School (2011). *DoD Army PPBE Executive Overview*. <http://acqnotes.com/Attachments/Army%20PPBE%20Executive%20Primer.pdf> (December 2011).
- Beck, K. et al. (2017). Manifesto for Agile Software Development. Accessed March 20, 2017. <http://www.agilemanifesto.org/>
- Cao, L. Ramesh, B. and Abdel-Hamid, T. (2010). Modeling Dynamics in Agile Software Development. *ACM Trans. MIS I*(1) 5:1-26
- Chichakly K. (2007). "Modeling Agile Development: When Is It Effective?" *Proc Int'l Conf. of the System Dynamics Society 2007*.
- Cocco, L., Mannaro, K., Concas, G. and Marchesi, M. (2011). "Simulating Kanban and Scrum vs. Waterfall with System Dynamics", *XP 2011*.
- Cooper, K. and Mullen, T. (1993), "Swords and Plowshares: The Rework Cycle of Defense and Commercial Software Development". *American Programmer* Vol 6, no. 5, (May 1993).
- Cooper, K. (1994). "The \$2,000 hour: how managers influence project performance through the rework cycle". *Project Management Journal*, 25(1), 11–24.
- Defense Science Board. (2009) "Department of Defense Policies and Procedures for the Acquisition of Information Technology" Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics: Washington, DC, (March 2009).
- Duvall, L. (2012). "Be Quick, Be Useable, Be On Time: Lessons in Agile Delivery of Defense Analytic Tools", Brookings Center for 21st Century Security and Intelligence Policy Paper, June 6, 2012
- Elbanna, A. and Sarker, S. (2015). "The Risks of Agile Software Development: Learning from Adopters." *IEEE Software* 33: 5, pp. 72-79 2015.
- Glaiel, F., Moulton, A. and Madnick, S. (2013). Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods. *Proc. 31st International Conference of the System Dynamics Society*, Cambridge, Massachusetts, USA -- July 21-25, 2013.
- Kong, X., Liu, L. and Chenc'a, J. (2011). "Modeling Agile Software Maintenance Process Using Analytical Theory of Project Investment", *Procedia Engineering*, Volume 24.
- Lapham, M. et al. (2010). *Considerations for Using Agile in DoD Acquisition*. Technical Note CMU/SEI-2010-TN - 002. Software Engineering Institute, Carnegie Mellon University.
- Leffingwell, D. (2017). *SAFe Reference Guide*. Scaled Agile, Inc.

- Little, J. D. C. (1961). "A Proof for the Queuing Formula: $L = \lambda W$ ". *Operations Research*. 9 (3): pp. 383–387. doi:10.1287/opre.9.3.383. JSTOR 167570.
- Madachy, R. J. (2008) *Software Process Dynamics*, John Wiley & Sons, Inc., Hoboken, NJ, USA
- Marjaie, S. and Rathod, U. (2011). "Communication in Agile Software Projects: Qualitative Analysis using Grounded Theory in System Dynamics", *Proc. Int'l Conf. of the System Dynamics Society 2011*.
- McKenna, N. A. (2005). Executing Major Projects through Contractors. *International Conference on System Dynamics 2005*.
- Miller, S. and Ward, D. (2016). *Update 2016: Considerations for Using Agile in DoD Acquisition*. Technical Note CMU/SEI-2016-TN-001, Software Engineering Institute, Carnegie Mellon University.
- Moulton, A., Tazyeen, F., Becker, D. & Madnick, S. (2015). A System Dynamics Exploration of Agency Theory Interpretations of ERP Acquisition Methods in DoD. *Proc. 33rd International Conference of the System Dynamics Society*, Cambridge, MA (July 19-23, 2015).
- Nicolette, D. (2016) Choosing an Agile Scaling Framework: A Handbook for Decision-Making. <http://leanpub.com/choosing-an-agile-scaling-framework>
- Reinertsen, D. G. (2009) Principles of Product Development Flow. Redondo Beach, CA: Celeritas. pp.73-75 and 148
- Roos, D., Womack, J., and Jones, D. (1991). The Machine That Changed the World : The Story of Lean Production. Harper Perennial, ISBN 0060974176, ISBN 978-0060974176
- Scaled Agile Framework (2016). "Scaled Agile Framework - Big Picture", [Online]. Available: <http://scaledagileframework.com/>.
- Scaled Agile Framework (2016). "SAFe 4.0 Introduction, Overview of the Scaled Agile Framework for Lean Software and Systems Engineering", July 2016. [Online]. Available: <http://www.scaledagileframework.com/introduction-to-safe/>.
- Tignor, W. (2009), "Agile Project Management", *Proc. Int'l Conf. of the System Dynamics Society 2009*.
- Tignor, W. (2011), "Agile Essence", *Proc. Int'l Conf. of the System Dynamics Society 2011*.
- Trammell, T., Moulton, A., and Madnick, S. (2016). Effects of Funding Fluctuations on Software Development: A System Dynamics Analysis. *Engineering Management Journal* 28:2.
- Version One (2016). "10th Annual State of Agile Report", <https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf>, 2016.

Appendix A. SAFe Big Picture (<http://www.scaledagileframework.com/>)

