Managing Discovered Scope Within Hybrid Agile Stage-Gate Project Delivery Systems

By

Thomas M. Johnson

B.S. Mechanical Engineering
University of Wisconsin – Madison, 2003

B.S. Agricultural Sciences
University of Wisconsin – Madison, 1995

SUBMITTED TO THE SYSTEM DESIGN AND MANAGEMENT PROGRAM IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN ENGINEERING AND MANAGEMENT
AT THE
MASSACHUESETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 2021

Signature of Author:_____
Department of Systems Design and Management
December 1, 2020

Certified by:_____
Eric Rebentisch
Research Associate
Department of Sociotechnical Systems Research Center

Accepted by:_____
Joan Rubin
Director, System Design and Management Program

Managing Discovered Scope Within Hybrid Agile Stage-Gate Project Delivery Systems

By

Thomas M. Johnson

Submitted to the Systems Design and Management Program on December 1, 2020 in Partial Fulfillment of the Requirements for the Degree of Engineering and Management

ABSTRACT

Complex mechatronic projects have machine functionality dependent upon substantial embedded software content delivered in coordination with the hardware componentry. This situation creates a dilemma for project leadership as they determine which methods to utilize for managing the project. One option is to utilize a hybrid approach where comingled Stage-Gate and Agile methods serve both hardware and software activities. However, the need for synchronized delivery schedules between the hardware and software components is not addressed well by Agile methods, which do not emphasize forward planning. In contrast, the uncertainty in defining software scope challenges the up-front scope definition relied upon by Stage-Gate methods.

Three independently operating project delivery systems have each spent more than ten years weaving Agile software development methods into the classic Stage-Gate approaches to make their hybrid project management systems. This study interviews Agile and Stage-Gate leadership roles within each of these three project delivery systems to identify what has evolved to keep the schedule expectations for scope delivery aligned to the discovery of additional scope while software development progresses.

This study finds that both the Stage-Gate and Agile leaders interviewed call for more work to be done in the project planning stage to improve the inclusion of more rigorous software scope identification activities. It also finds several differences in the design stage activities across the groups studied concerning how they accommodate the discovery of new software scope into the overall scope and schedule expectations for the project, each with a differing level of effectiveness.

The most effective traits include the formalized identification and capture of the product decomposition and architecture so that it can be used to estimate software scope, schedule, and resources more accurately upfront in the planning stage. During the design stage, the most effective project delivery systems leverage the cultural acknowledgment and leadership's enforcement of the stakeholders' need to adjust their scope expectations in response to new scope discoveries. The addition of repeating two-month planning events deliver timely forecasts of software deliveries, and frequent scope management meetings allow for rapid adjustment to software scope discoveries. Each software delivery system added dedicated Software Delivery Lead roles to act as the liaison between the Agile and Stage-Gate management methods and to formulate mitigation activities with the rest of the functional area leads to the mechatronic product.

Project delivery system developers may use these findings as a set of lessons learned to guide their pursuits with the integration of Agile practices into an existing Stage-Gate process. Others could build upon these findings by repeating the activity with other case studies to see if a pattern emerges, which could guide the creation of more specific Agile Stage-Gate frameworks.

Thesis Supervisor: Eric Rebentisch
Title: Research Associate

Contents

Table of Figures

Definition of terms

Project Delivery System: The sociotechnical system, including the people, processes and tools built up around the mission to deliver a product to the marketplace.

Mechatronic Product: A product that is built of both mechanical and electronic componentry for the purpose of completing mechanical tasks. Typically, the electronic content is present to provide the means for controlling the mechanical aspects of the product while it executes its mission.

Embedded Software: The software portion of the mechatronic system. The software is "embedded" within the system and purpose built for that specific device. Contrast this with application software on a personal computer, where the software is built and shipped separately from the hardware and is agnostic to any specific hardware device.

Software Delivery Teams: The groups responsible for delivering the embedded software content of the mechatronic system. For this study, the software delivery teams are utilizing Agile principles to manage their team and develop the product.

Project Leadership, or **Project Management**: While there may be leadership and management roles specific to the software delivery organizations, this study will use these terms to represent the leaders and managers of the overall mechatronic project.

Agile Project Leadership: Leadership roles pertaining to the development, management and execution of the Agile methods for the software delivery teams.

Scope: The tangible artifacts that developers create as part of product development activities. It includes the final deliverables, as well as any intermediate artifacts that are created for the purpose of creating the product or managing the project. Refer to Moser [3] for a more complete definition.

Discovered Scope: Scope that is not identified or known to be needed to meet the product's objectives until development has begun. It is sometime referred to as "Unknown Unknowns." Since it is not understood up-front, it is a disturbance to the project plan while the project is underway.

Expected Performance: What the project leadership perceives as the thresholds for the delivery schedule, cost, and quality that allow them to determine if the software delivery teams are doing a "good job".

Actual Performance: the actual performance of the software delivery teams with respect to the schedule, cost, and quality of the delivered scope.

Expectations gap: The difference between the Expected Performance and the Actual Performance. This may be explicit and measured, or implicit and residing within the minds of the project Leadership.

1. Purpose

1.1 Motivation:

Agile methodologies gained popularity in the early 2000s following the "Agile Manifesto," published in 2001 [1]. The philosophy resonated with software developers and organizations developing software products. Since the Agile Manifesto publishing, a large industry has built up around providing consulting, tools, published books, how-to guides, and other products and services.

The original Agile methods grew from the business scenario of a single team doing development for a given customer. Those involved with the original agile methods designed them to utilize direct customer contact with the team. The customer frequently reviews the product deliveries and provides acceptance or guidance for corrections in the moment.

A challenge with the original classic Agile methods has been applying the methods to organizations with many dozens to many hundreds of people working on a single project simultaneously. In this situation, it is no longer practical to have the actual customer review results from each of the many teams involved. Agile framework developers have added provisions for representing the customer within each team. These provisions effectively become the project management activity of the project within the organization. In addition, large projects spread across multiple teams requires the results of their efforts to contribute to the emergence of not only a desirable product but at an acceptable cost and schedule.

Another dimension to the challenge of managing software delivery projects occurs when the delivered product is a mechatronic system. Mechatronic systems have embedded software controlling the system's mechanical devices. This scenario creates dependencies between Stage-Gate managed hardware development and Agile software development. How do the additional dependencies supporting these mechatronic systems' design needs get handled within the Agile project management strategies?

Project managers face a dilemma when leading the delivery of complex electromechanical systems with substantial electronic software content. Planning and managing the project with classic Stage-Gate approaches have been a proven method to support mechanical hardware's project management needs. However, many will argue that software development is better supported using "Agile" methods that favor incremental delivery flow-based process control. Project planning is accomplished differently within these two paradigms. Stage-Gate methods expect the planning to occur up-front and the scope to remain stable. Agile frameworks provide team-level tactical planning of small tasks. Agile expects small work units to be delivered to the team, leaving the project decomposition and prioritization of the larger system to be done by the customer. Neither paradigm provides for a path to support the other. Nevertheless, we often see organizations attempting to utilize both approaches within a single project, having to create the mechanism and behaviors to force some form of effective exchange and coordination between them.

This study seeks to identify what mechatronic product delivery system leaders create to align the scope and schedule expectations when developers discover additional needed scope as the project progresses. It will also identify if structural, procedural, and behavioral attributes utilized by the project delivery system leads to what leadership considers successful software delivery within these hybrid organizations. This study will accomplish this by investigating several project delivery systems that have stable and long-running experience with operating in Agile frameworks. The study will use observational data, interviews, and surveys to quantify the findings. The results will assist those who are looking to improve the methods to manage projects within organizations using Agile methodologies so that they may be more effective in the delivery of successful projects.

1.2 System of Interest

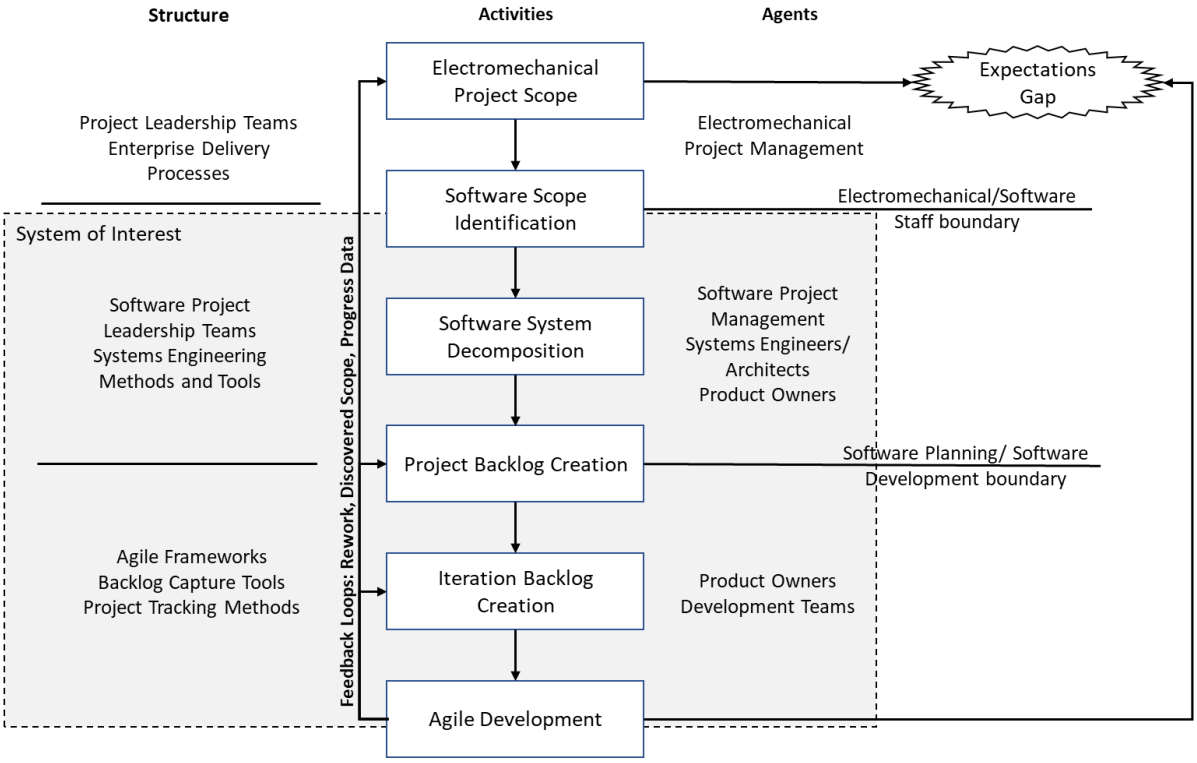


Figure 1: Typical Activities, Agents and Structures for Managing Agile Software Development

The system of interest for this study is the project management activities that support development teams creating software that controls large, complex electromechanical systems. To assist in understanding the problem statement, consider the situation in Figure 1, which summarizes the typical sequence of activities for software development within the project delivery system and the interfaces to the leadership of the mechatronic product development project. To the left of the activities are some of the structural elements related to this system, and to the right are the typical agents executing the activities. This system is part of a larger sociotechnical system, with many development teams working

collectively to deliver the overall software solution. Each development team supports one or more development programs that are ongoing at a given time. The leadership at the management level of the overall mechatronic product uses a traditional Stage-Gate approach for its project management. The software development organizations developing solutions for the mechatronic projects have adopted Agile methodologies (discussed below) many years ago using outside consultants and formal training. Thus, the development teams operate in a stable organizational culture that expects the use of the prescribed methods and ceremonies that they consider part of its adoption of Agile.

The following introduces the typical activities related to delivering on a project, as reflected in Figure 1.

Identify Requested Software Scope:
The leadership of the mechatronic project completes this activity. The deliverable is documented expectations of what is to be delivered by the organization's software development part. Many things impact the effort the electromechanical project planners apply to the activity, including the complexity of the software functionality asked for and the specificity of the performance and quality captured in the request.

Software System Decomposition:
Developers need to decompose the large, complex systems into a set of smaller activities so that many teams can work in parallel. The amount of effort applied, the time of completion (e.g., up-front planning, or later during the development activities), and the decomposed units' size and specificity impact the effort involved with this activity.

Product and Team Backlog Creation:
This activity identifies the work assignments, known as 'stories' and the deliverables of the story. Stories are small in scope, usually of a size where the design, build, and test activities for delivering the scope can be completed in a consistent, fixed amount of time, usually, 1 to 4 weeks depending on the Agile framework. Stories are prioritized within a product backlog where the entire scope of the project is captured and prioritized so that developers deliver the most critical work first. Each story is assigned to a team and will be pulled into the iteration backlog when it has the capacity for it. Agile frameworks allow for teams to have the right to reject stories if they are not sufficiently defined. However, leadership expects the teams to adhere to the assigned priority for determining what to pull in next. Project effectiveness is altered depending upon the effectiveness of establishing and delivering the highest priority work first, the clarity of the expectations for the stories, the effort applied by the project owner and the developers to create the backlogs, and the ability for the plans built into these backlogs to remove/manage dependencies.

Agile Development:
Agile development includes all the design, build, and test activities required to deliver the scope incrementally. While a discussion of these items is beyond this study's scope, there are several items related to project planning and management. This includes dependency handling, effort allocation across multiple projects, and the overall number of stories in progress at a given time.

Software development teams will often find that the planned effort for delivering a story will take longer than what was initially estimated. They may also find they need to deliver more things than initially planned to deliver upon the project's functional expectations. We will refer to this as discovered scope, and it presents a challenge to project management because additional scope requires additional time (if

resources are fixed). However, the mechanical systems are relying on the software to be available at a specific date.

Feedback Loops:

The Agile development activity provides feedback for how the teams are progressing on the stories. They also identify missed scope and new scope opportunities. Discovered errors requiring rework are added back into the backlogs, and risk information is relayed to leadership. Organizations differ in how they respond to this feedback, and this can impact the project's success. These differences include the effectiveness the feedback has on managing the expectations of the project leadership (thus changing the expected scope and date), the ability for the organization to re-plan, managing the influx of new scope, the number of planning resources available, and the ability to apply more resources when needed.

One challenge with hybrid project management methods is in defining success. See Figure 2. Stage-Gate approaches typically define the project's scope, then estimate the time and cost of the project, and then compare the actual scope, cost, and schedule to the budgeted predictions. Agile methods start with fixed resources and then focus on developing the most important things first. When the delivery date arrives, an Agile system will deliver a valuable product, regardless of how much actual scope is present. If deemed valuable, developers pursue a second release with the next batch of the highest priority scope. This study will discover how these hybrid project management methods define success where success is the level of satisfaction the leadership has with the scope, schedule, and cost of the delivered software content.

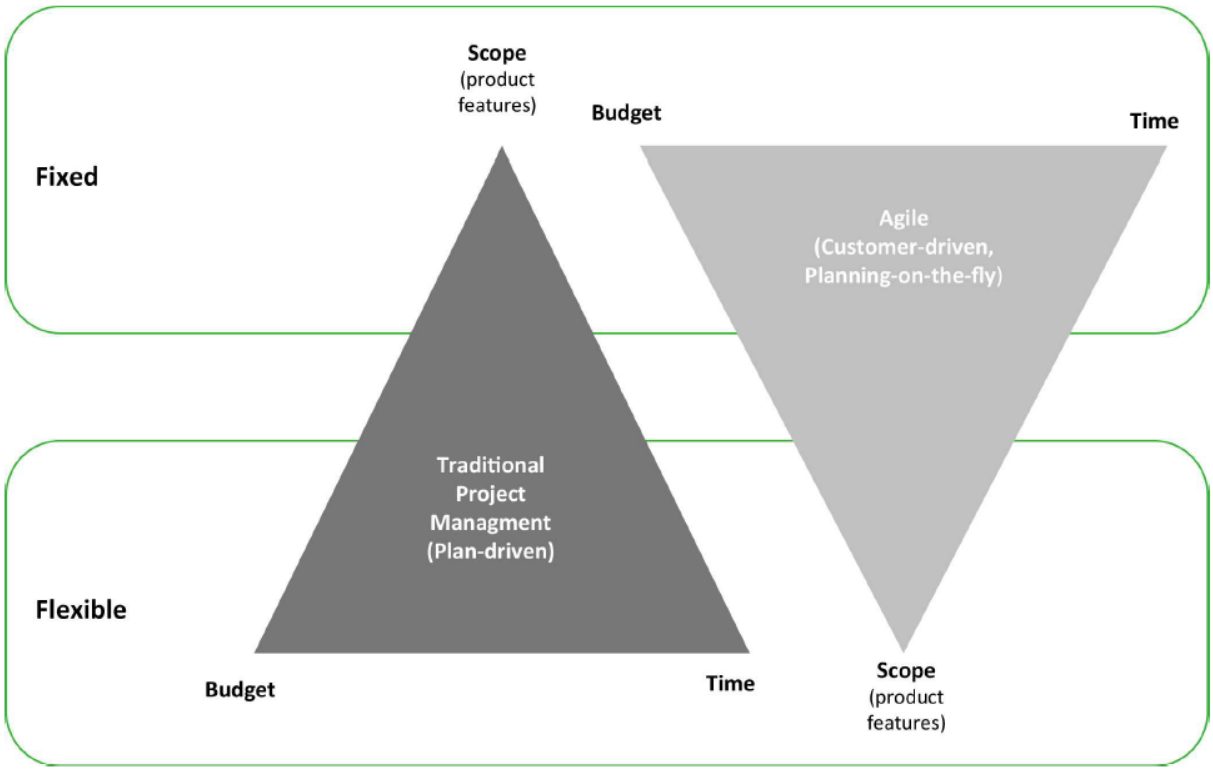


Figure 2: Fixed vs. Flexible Attributes of Agile and Stage-Gate Project Management Strategies [2]

The project delivery system's structural attributes include the formal and informal activities and roles that exist for project management purposes and the formal and informal organizational reporting structures. It includes the following:

- The presence of any leadership teams, development teams and the responsibilities of those teams.
- The alignment of individuals or teams to specific activities and allocating of specialists into the team. The attributes of the solution itself, such as the relative size and complexity of the scope and how interdependent it is with the mechatronic hardware.
- The design tools, management tools, and Information Technology infrastructure available to the teams.

Procedural attributes are the formal and informal processes and activities executed by the agents filling the roles. This includes the following:

- The processes used for formulating and delivering the requests for scope from the software delivery teams. The processing and decomposition of those requests into actionable tasks by the software delivery teams.
- The methods used to monitor delivery progress and quality and the processes that enable steering and managing the software development teams.
- The formal communication and coordination activities, such as meetings and reviews, within and between teams.
- The documented processes and frameworks used to guide the project execution and timing of the activities mentioned above.

Behavioral attributes include the skill sets, capabilities, culture, and social behaviors that the agents utilize while completing the activities. It includes the following:

- The amount and quality of informal communication that the agents complete and the establishment of relationships, networks, trust, and rapport that the agents create to manage the project.
- The skill with defining and communicating the expectations of the tasks, the ability to focus the attention of the team(s), and to manage project scope.
- The ability to receive and deliver both good and bad news as feedback and work together as a team.
- The amount of effort that the agents put into the activities and actions that they take.

System of Interest = Management System for the Software Development of Complex Electromechanical Products		
Emergent Property of Interest = Perceived Success with the Software Delivery		
Attributes Influencing Success		
Structural attributes	Procedural Attributes	Behavioral Attributes
Role definitions	Work execution methods	Communication skills
Reporting structures	Methods for identifying scope	Relationship building and management
Team organizational structures	Processes for decomposing the request.	Networking
Distribution of skillsets	Project monitoring methods	Trust and rapport with leadership
Size and complexity of the requested software system	Communication activities	Expectations management skills
Team locations	Coordination activities	Scope management
Team co-location	Project estimation methods	Attention management (of self and others)
Communications infrastructure	Process documentation and/or frameworks	Effort applied to each of these activities
Types of design, management tools available	Relative execution timing of the above activities	
Existence and availability of support teams	Prioritization and attention allocation activities	

This paper will first summarize the attributes of software that challenge classic project management techniques. It then briefly reviews traditional Stage-Gate frameworks, three classic Agile frameworks, and newer frameworks that scale the classic Agile methods to large organizations. Then it reviews hybrid systems that combine Agile and Stage-Gate frameworks. A method for assessing a project delivery system's completeness is proposed based upon System Dynamics principles and the best practices identified within the Project Management Book of Knowledge [2]. An experimental design is discussed which intends to identify the techniques used and the effectiveness of those techniques in managing the ongoing discovery of required project scope. The paper then closes with conclusions and possible next steps.

2 Literature Review

2.1 The Nature of Software:

Using software as a medium for making solutions has several inherit attributes that cause project management challenges. Leffingwell [4] discusses several of these attributes and how they are not addressed by classic project management practices. Each of these items is expanded upon below.

1. It is not possible to define accurate requirements for software projects upfront.
2. Requirements for software products change rapidly, causing the initial capture to become quickly outdated.
3. System integration, when completed at the end of the development activities will not go well and will drive a large amount of rework.
4. Software innovation activities cannot be predicted.
5. Software is easy to refactor.

As a rationale for item one, Leffingwell [4] asserts that, as opposed to mechanical systems, software is intangible. Thus, one does not have the luxury of drawing a picture of what is wanted like for a mechanical product. Therefore, even though the customer and the developer may go through a rigorous effort to explain and document what is required, the customer and developer will still not have an identical understanding of what is expected. In addition, the intangible nature of software increases the likelihood that the customer may not realize that what was asked for. Thus, the requirements do not match what was wanted. Once the product is delivered, the customer will realize the shortcomings in what was asked for and therefore change the requirements to match what was really wanted.

As a final rational for item one, Leffingwell [4] asserts from his experience that when software solutions are delivered and the customer begins to use it, the customer will change their behavior in response to the new software and then want something different. Effectively, Leffingwell is calling out the fact that the customer and software are part of a *Sociotechnical System*, where the behavior of the agents of a system and the attributes of the instruments of the system are simultaneously changed by the influence of the other.

The rationale for item two is based upon an assertion, likely coming from his experiences, that the expectations of software products rapidly change, and that this change is so rapid, that the needs of the product will change substantially even during the time that the product is being developed. He is effectively calling out that products using software solutions are evolving so rapidly that the classic, up-front planning approaches of classic project management methods cannot keep up. Any requirements that are captured during up-front planning are likely to become outdated as the project progresses.

Item three ties back to the challenges with working with a medium that is intangible and pliable in nature. This leads to the inability to explicitly manage the evolution of the interfaces being created and modified by many teams working in parallel and results in integration issues to be created, but not be identified until the end of the project when the integration work actually occurs. This ultimately causes the integration activities to take much longer than predicted.

With item four, Leffingwell [4] calls out two uncertainties that exist during the planning phase of a project which lead to the inability to estimate the time it will take to create software

- Unknown Unknowns: the existence of things to do that are not identified during planning. When it is identified during the development, we refer to it as discovered scope
- We are not good at predicting the actual time it will take to develop the tasks that we know about.

Leffingwell [4] asserts from his personal analysis and studies that software teams will underestimate how long it will take to develop a given software artifact by at least a factor of two. He claims that the sources of error in the estimation come from two sources; 1) assuming that there is no rework, and 2) software development is innovative by nature, and therefore will have discovery activities that are uncertain in length.

This is consistent with the observations of Brooks, 1995 [5] regarding the struggles to predict the actual completion time and effort associated with software development projects. System dynamics principles have been used by many [Lyneis and Ford, 2007 [6]] to quantify the emergent properties of a project delivery system when initial scope and schedule estimations are not accurate. Januszek, 2017 [7] attempted to explain the impact of unknown unknowns by extending the use of systems dynamics to explain an Agile project management system. These efforts will be discussed with more detail in a later section

Item five is essentially identifying that writing lines of code is cheap and fast. It is not like mechanical hardware where there are extensive costs in procuring the materials, fixtures, machining activities, etc. and all these activities take large amounts of time to procure and execute. Software developers can replace lines of code in seconds, and programming tools that enable this to occur quickly are relatively cheap. This is a key attribute of software that enables many of the Agile design principles.

2.2 Other Causes that Decrease Delivery Performance

Developers and development teams have a finite amount of effort available for a given unit of time. They are also faced with many competing requests for the application of that effort. The following paragraphs illustrate additional challenges that are difficult to account for with up-front project planning.

Innovation vs Delivery: As discussed above, software development is innovative by nature. Thus, the development teams engaged in creating software are “innovation teams” as discussed and summarized by Thayer et. al, 2018 [8]. Innovation teams have the responsibility to identify new and innovative ways to solve problems, ie. “thinking”. They are also responsible for delivering these new solutions to production, ie. “doing”. Thus, the team faces a paradox in that it must possess the skillset for innovation and delivery and manage the allocation of time and attention to each of these activities. Since software is so cheap and easy to change, one can conclude that software development teams have an extra level of challenge with the transition between innovating and delivering because of the relative ease to try new ideas with little need to coordinate with others in the moment. It is easy to stay within the innovative work activities at the expense of delivery.

Multiple Projects: Some organizations assign product teams to single projects, while other organizations have a team working on many projects at once. The Project Management Institute [2017] [2]summarizes this dependency using Figure 3.

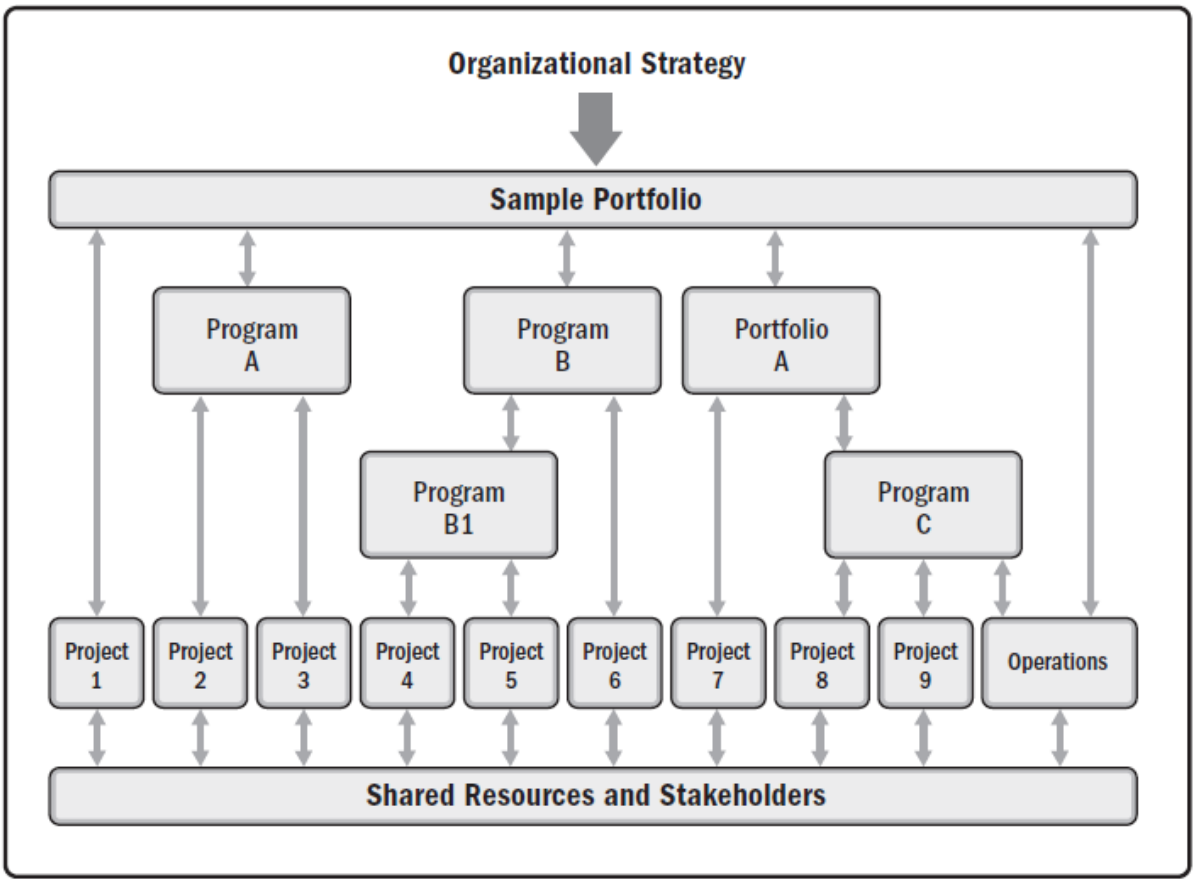


Figure 3: Strategic Sharing of Resources Across Multiple Projects [8]

Project Management Efforts: As is often part of project management best practices, development teams typically will have some responsibility for tracking and reporting progress on the project(s) that they are working on. This results in effort consumption in the form of written status reports, status meetings, and/or participation in project management/steering committees.

Coordination: When the project is large and complex, the scope of the project becomes larger than what can be completed by a single development team, thus the project will be decomposed into smaller activities that deliver pieces of the overall system. Multiple teams will then be formed to work on the different activities in parallel. With this decomposition and team allocation comes the introduction of interfaces between different elements of the product and the product teams that are engaged. Effort must be applied to manage these interfaces and coordinate the efforts of the many teams. Moser and Wood, 2015, [9] discuss coordination as an activity that it has no direct product, it is not performed alone, and it often is done unconsciously. They continue by adding that complex engineering projects

and the trend of having teams distributed add to the effort to coordinate and then offer a model which reflects the coordination as a co-dependence between teams. Should one or both teams fail to allocate sufficient attention to the need for coordination, progress will slow, quality will decrease, and/or rework can be required.

The research of Galbraith [10] further demonstrates the need to apply effort towards coordination activities. He proposes a model that describes the organizational response to the need for coordination as a need to process information. He then describes several information processing constructs that allow for the necessary decision making and action plans to be formed which allow for effective coordination [Figure 4.

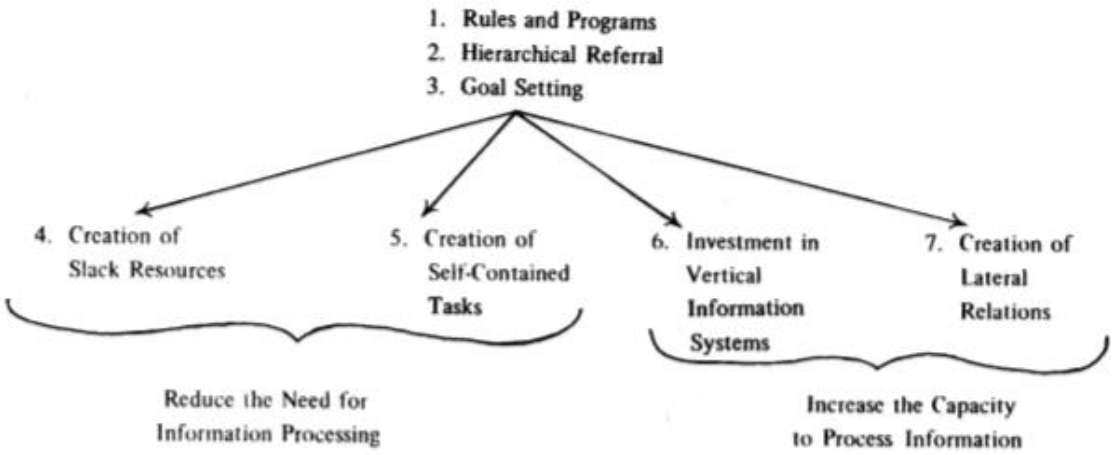


Figure 4: A hierarchical progression to the application of strategies for supporting team coordination [10]

Knowledge Creation and Transfer: Maintaining and improving the knowledge within the organization is an important activity for the sustainment of organizations engaged in new product development. Nonaka, et. al. describe the need for companies, and thus the teams and individuals within the company to allocate effort to two categories of activity required for knowledge creation, exchange, and management: SECI, the creation, capture, and transfer of knowledge, and Ba, the creation and maintenance of an environment that facilitates the SECI activities [Figure 5]. These are continuous and dynamic activities that compete for attention with everything else that a development team must do.

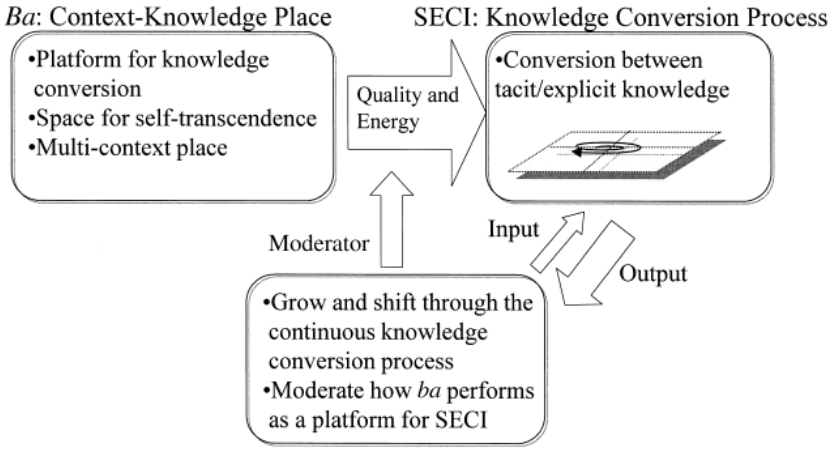


Figure 5: The SECI-Ba Model for Explaining Knowledge Generation

2.3 Project Management Best Practices

The *Project Management Book of Knowledge (PMBOK)* [2] provides a general, overall discussion of the best practices for setting up and managing projects. It asserts that project management as a discipline helps organizations to:

- Meet business needs
- Satisfy stakeholder expectations
- Be more predictable
- Increase chances of success
- Deliver the right Products at the right time
- Resolve problems and issues
- Respond to risks in a timely manner
- Optimize the use of organizational resources
- Identify, recover, or terminate failing projects
- Manage constraints (scope, quality, schedule, costs, resources)
- Balance the influence of constraints on the project
- Manage change in a better manner

It continues with the assertion that project management should be considered a strategic competency within organizations because of its ability to:

- Tie project results to business goals
- Compete more effectively in their markets
- Sustain the organization

- Respond to the impact of business environment changes by appropriately adjusting project management plans.

A major portion of PMBOK is dedicated to the creation of the "Project Management Plan". It defines this as "the document that describes how the project will be executed, monitored and controlled, and closed". After applying a *Systems Thinking* perspective to this definition, one would conclude that the project management plan is the creation of the *Project Delivery System* that will be used to deliver the desired product or service.

PMBOK categorizes the activities that are part of effective Project Management Plans into "Project Management Process Groups" and "Project Management Knowledge Areas". See Figure 6. The activities themselves are the project management best practices that should exist within a Project System. The book states that the Project Management Plan, should include these activities unless there is specific reason to scale them out. While the legacy of PMBOK is based upon Stage-Gate frameworks, the book asserts that these activities are important for any project system, no matter if it is based upon Stage-Gate, Agile, iterative, or any combination of these.

Knowledge Areas	Project Management Process Groups				
	Initiating Process Group	Planning Process Group	Executing Process Group	Monitoring and Controlling Process Group	Closing Process Group
4. Project Integration Management	4.1 Develop Project Charter	4.2 Develop Project Management Plan	4.3 Direct and Manage Project Work 4.4 Manage Project Knowledge	4.5 Monitor and Control Project Work 4.6 Perform Integrated Change Control	4.7 Close Project or Phase
5. Project Scope Management		5.1 Plan Scope Management 5.2 Collect Requirements 5.3 Define Scope 5.4 Create WBS		5.5 Validate Scope 5.6 Control Scope	
6. Project Schedule Management		6.1 Plan Schedule Management 6.2 Define Activities 6.3 Sequence Activities 6.4 Estimate Activity Durations 6.5 Develop Schedule		6.6 Control Schedule	
7. Project Cost Management		7.1 Plan Cost Management 7.2 Estimate Costs 7.3 Determine Budget		7.4 Control Costs	
8. Project Quality Management		8.1 Plan Quality Management	8.2 Manage Quality	8.3 Control Quality	
9. Project Resource Management		9.1 Plan Resource Management 9.2 Estimate Activity Resources	9.3 Acquire Resources 9.4 Develop Team 9.5 Manage Team	9.6 Control Resources	
10. Project Communications Management		10.1 Plan Communications Management	10.2 Manage Communications	10.3 Monitor Communications	
11. Project Risk Management		11.1 Plan Risk Management 11.2 Identify Risks 11.3 Perform Qualitative Risk Analysis 11.4 Perform Quantitative Risk Analysis 11.5 Plan Risk Responses	11.6 Implement Risk Responses	11.7 Monitor Risks	
12. Project Procurement Management		12.1 Plan Procurement Management	12.2 Conduct Procurements	12.3 Control Procurements	
13. Project Stakeholder Management	13.1 Identify Stakeholders	13.2 Plan Stakeholder Engagement	13.3 Manage Stakeholder Engagement	13.4 Monitor Stakeholder Engagement	

Figure 6:Summary of Project Management Activities [10]

2.4 Stage-Gate Frameworks

Stage-Gate frameworks have been utilized for decades and are quite mature in their definition and use. They get their name from their typical pattern of activities (stages) followed by a leadership review (gates) which grants permission for the project to continue to the next stage. They are built upon the notion that the cost of correcting errors increases by orders of magnitude during the life of the project, and therefore, applying extra effort up-front to thoroughly plan and specify the expectations and deliverables of the entire project prior to any development activities results in improved project performance in the dimensions of scope, cost, schedule, quality, etc.

Stage-Gate frameworks tend to complete a sequential set of activities on the entire breadth of t project within each stage prior to beginning the work in the next stage. The work within any given stage provides the knowledge needed to decide if the project is still worth-while to pursue. Approval to begin the work in the next stage is granted after a leadership review, the “Gate” of the Stage-Gate framework. The stages are typically defined to align with the product lifecycle, which typically include stages for concepts, development, building/manufacturing, operation, and disposal. Several Stage-Gate frameworks, as summarized by Forsberg, K., H. Mooz, and H. Cotterman, 2005, [11] are provided in Figure 7.

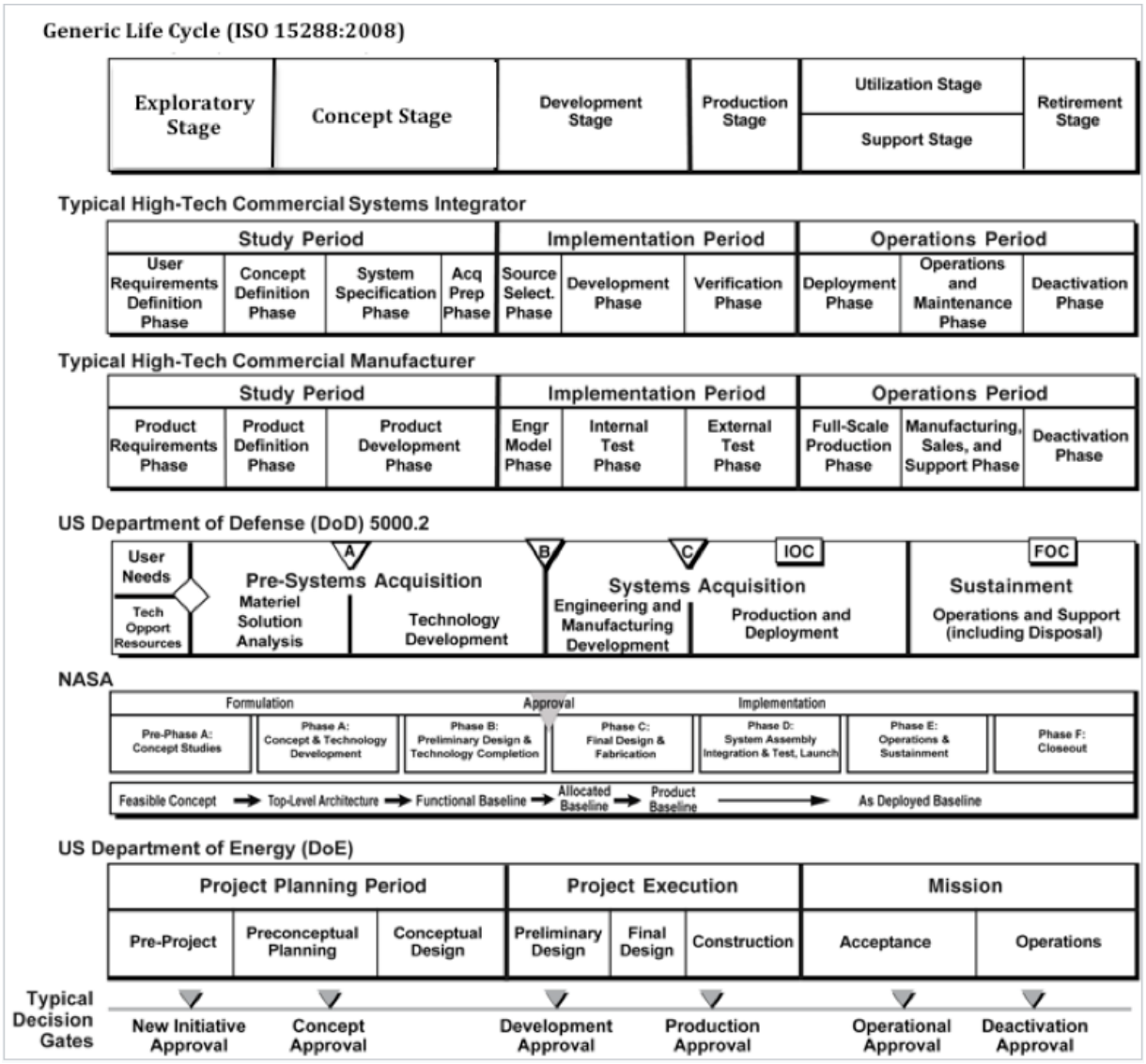


Figure 7: Summary of Various Stage-Gate Representations [11]

Complex engineering projects are expensive to execute [12]. They also tend to accrue most of their expenses in the latter stages of the project [Figure 8]. Stage-Gate frameworks are built to reduce the overall cost of the project by preventing expensive rework cycles in the latter stages of the project. The framework addresses this with using the notion that everything about the customer expectations of the product can be identified and fixed up-front through analysis and simulation. Stage-Gate frameworks often utilize systems engineering practices such as stakeholder analysis, decomposition, architecture,

requirements capture, and simulation activities in the early stages to minimize the opportunities for errors that would cause re-work in later stages.

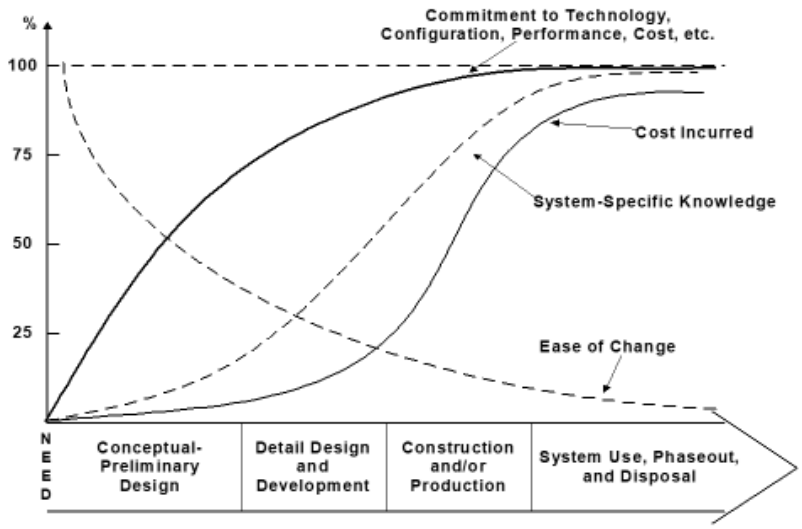


Figure 8: Typical Product Lifecyle Cost Commitments [12]

Stage-Gate frameworks are built upon an underlying assumption that there is nothing of value to deliver until the entire span of the specified scope is deliverable. The stages progress everything about the project along at the same rate until it is all ready for the next stage. This most basic form is therefore quite reliant on the up-front planning to be correct and that the surprises at the end are minimal. Some applications of Stage-Gate frameworks include iterative deployments for testing and validation purposes. These product delivery iterations allow for learning from the overall integration steps to be acted upon within the project by planning for additional design cycles. Further evolution of the frameworks yielded the concepts of the Vee-Model [13] where the project scope is broken down into subsystems and then components. See Figure 9. Then design on the components is followed by hierarchical layers of validation and verification of the components, then the subsystems, and then the overall system. Each layer of testing enables a rework loop to be taken on a smaller part of the system before advancing to the next larger part of the system. A continuation on this notion of iterative development is the spiral model [14], where the stages are repeated multiple times to yield prototypes that allow for testing and learning prior to a final iteration to deliver a completed product. See Figure 10.

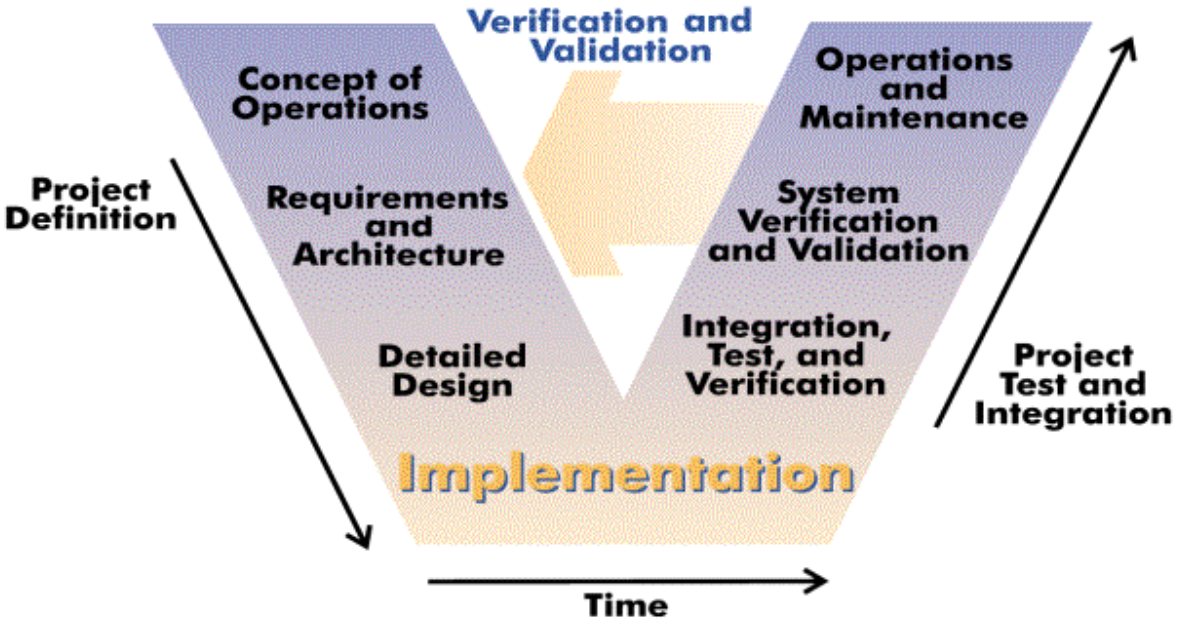


Figure 9: The "Vee Diagram" Summarized [13]

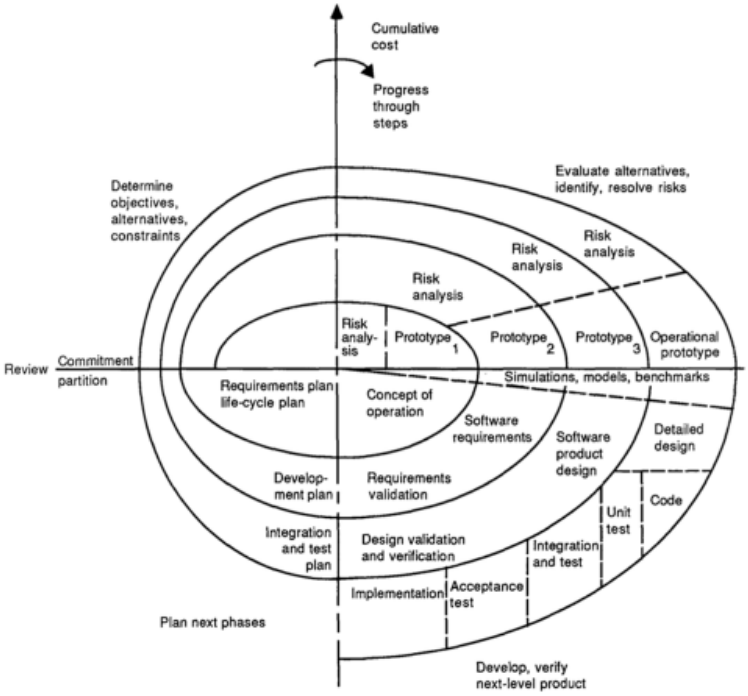


Figure 10: The Spiral Model [14]

2.5 Agile Manifesto

In light of the attributes and challenges with software development discussed above, a group of influential software development practitioners met in 2001 in Utah and released what has been titled "The Agile Manifesto" [1].

Figure 11: The Agile Manifesto [1]

This declaration has served as the vision that has guided the exploration and development of new management models for organizations engaged in software development. From this declaration came the following list of guiding principles [15], that provide more clarity to the statements in the Agile Manifesto:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

To summarize these principles in terms of scope, schedule and cost, Agile principles call for additional scope in the form of more integration and build events, as well as more release events and customer evaluations. This extra work is offset by a reduction in the effort required for documented requirements, planning and the overall troubleshooting of integration problems. It also implies an assumption that the scope can be delivered incrementally, and these incremental deliveries have value, even though the rest of the identified scope is not complete.

2.6 Scrum

Scrum is one of the major frameworks utilized by teams to embrace agile practices. The framework is attributed to and first published by Ken Schwaber and Jeff Sutherland in 1995, known as "The Scrum Guide" [16]. The framework is summarized in Figure 12 below [17]. As a framework, it identifies a fundamental structure, but stops short of prescribing detailed, step-by-step processes for the structural elements of the framework.

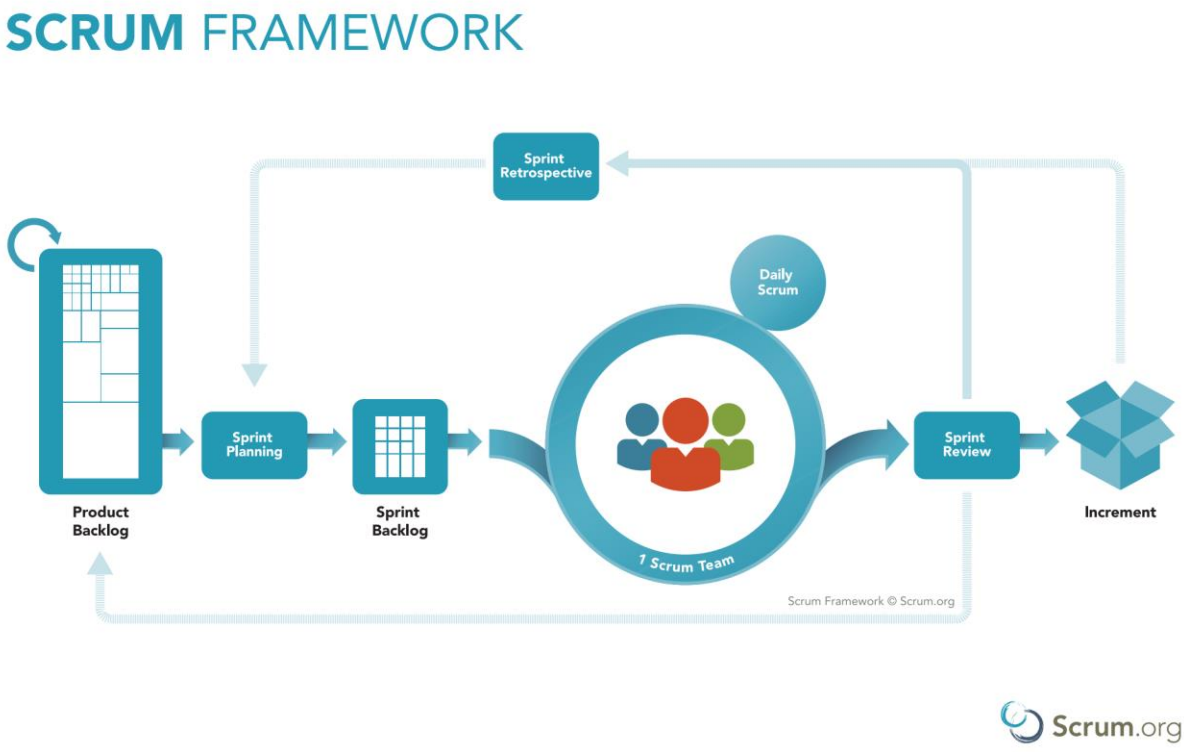


Figure 12: The Scrum Framework [17]

Schwaber and Sutherland explicitly call out empirical process and control theory, or empiricism as the theoretical guide for this framework. They specifically identify transparency, inspection, and adaptation

as key attributes of empiricism that are enabled by Scrum. The framework itself delivers upon these attributes using backlogs, sprint planning, daily scrums, and the sprint review.

Backlogs serve as the work breakdown structure in Scrum. There are two backlogs in the Scrum framework, the product backlog and the sprint backlog. The product backlog serves as the prioritized "to-do" list for the overall product. It is created and maintained by the product owner, who has sole discretion as to the content and priority of the items listed. The sprint backlog is the "to-do" list for the team for the next incremental unit of time (a "sprint"). In Scrum, sprints set the delivery cadence. It is expected that incremental additions or improvements to the product are delivered with each sprint. The sprint backlog is the scope that the scrum team agrees to deliver in the next sprint. The sprint backlog is built during a "sprint planning" event, where the team takes items from the product backlog into the sprint backlog. Each item is decomposed as needed and additional action items required to deliver upon the backlog items are added. In effect, the backlog serves as the both the list of deliverables (Scope) and the team plan (work breakdown structure) for the sprint.

There are three key activities that provide the feedback mechanisms for monitoring project delivery progress and team performance, The daily scrum, the sprint review, and the sprint retrospective. The daily scrum is a deliberately short meeting where team members identify what has been done, what will be done that day, and what help they need. Data and knowledge from this meeting are intended to allow for the team to adjustments to the work plan (sprint backlog). The product owner also participates and monitors for information that may cause the need for adjustments to the product backlog. Sprint reviews exist to evaluate the deliverables against the needs of the project. If the deliverables are not deemed to meet customer expectations, due to improper execution, improper definition of what was needed, or because the design concept was inadequate, the information will be used to add to or change the product backlog. The information may also feed into improving the processes that the team uses to deliver. The sprint retrospective is an assessment of the effectiveness of team execution. Items identified during this meeting are used to determine actions that will improve team performance.

2.7 Kanban

David J. Anderson is credited as the creator of the Kanban method for software development [18]. Kanban is a methodology that takes in considerable influence from Lean manufacturing principles which aim to reduce inefficiency, waste, and variability from manufacturing processes. Kanban pursues these same goals as its primary mission.

The Framework of the Kanban methodology models the typical activities a given team uses to deliver on the requested functionality. It is a flow model where the requests are tracked through the different work activities that the team completes to deliver upon the request. One could use an analogy of the layout of a factory floorplan, where a request for a product comes in on one end of the factory, and then all of the activities are visible as machines that complete the manufacturing activities along the way and then the product comes out the other end. These are called "flow models" because the requests come in and products come out continuously, with a number of requests "in process" and working their way through the various activities. Kanban methods often depict these models with the use of a "kanban board" as illustrated in Figure 13 below.

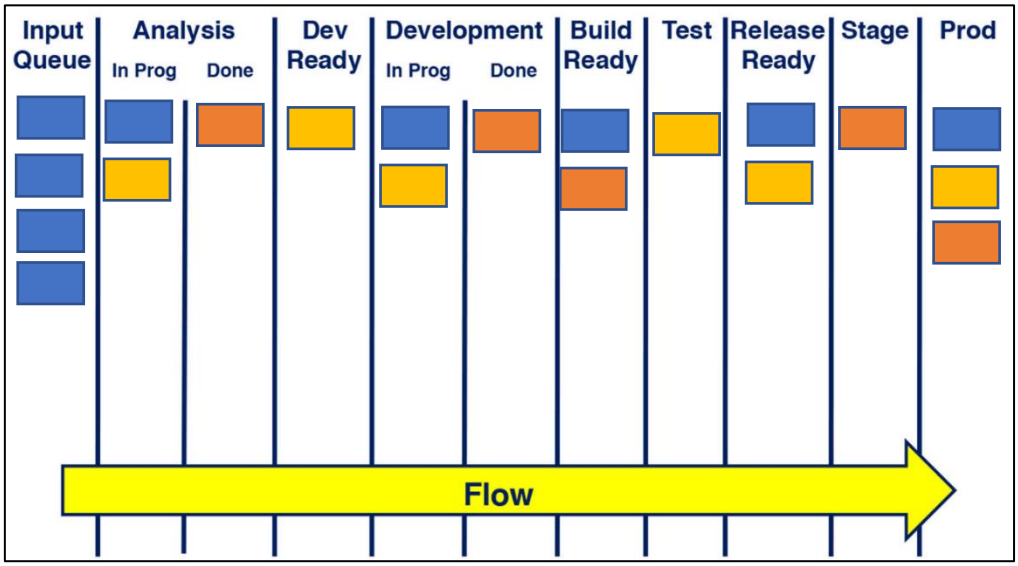


Figure 13: Example Kanban Chart

Like Scrum, Kanban is looking to bring structure to the development process so that it can be measured, and therefore, improved upon. Kanban aims to identify predictability by driving for consistency in the time it takes for a request to be processed through the team's development activities. The flow model allows for process bottlenecks and blocked work to be readily identified so that remedies can be found. This creates mechanisms for continuous improvement of the development activities and thus ever-increasing productivity.

In addition to the modeling and monitoring of the flow of requests through the team activities, Kanban reinforces a culture that focuses upon efficient delivery with the use of several metrics that focus on measuring consistency and throughput speed and through the limitation of "Work in Progress". A work in progress limit constrains the number of requests that can be within the team's workload at a given time. To work on the next request, one of the earlier requests needs to be completed and expelled from the team's workflow. This drives the team to focus on completing requests instead of allowing them to reside in an incomplete state for long periods of time.

The work in progress limit also structures the interface to project management activities. Strictly speaking, the framework of a Kanban model only contains a provision for a short list of requests, known as an input queue, that are ready for the team to pull-in to their workflow system when their work in

progress limit allows. Any prioritization decisions regarding what is placed into the input queue are handled by project management activities outside of the workflow model and outside of the team. The decisions as to what the team pulls from the input queue is prescribed by rules, such as a First-In-First-Out approach, or some type of prescribed priority system where project management activities assign priority classes to the requests and the team pulls-in requests based upon these classes.

2.8 Extreme Programming (XP)

Kent Beck conceived the Extreme Programming (XP) concept in the mid 1990's. His work was first published as a book in 1999, and a second edition was released in 2005 [19]. XP intends to improve the efficiency and quality of software development by having teams change their methods and practices for creating software. Beck identifies "Values" that are at the philosophical core of the XP concept. Then, "Principles" begin to focus the value statements to more tangible, actionable ideas. Finally, "Practices" prescribe specific execution practices that teams employ. These are summarized in Table 1 below. Like Scrum and Kanban, the primary focus is on execution, but there is some discussion about project planning and management and is summarized in below. A detailed review of all these items is beyond the scope of this paper and will be left as an exercise for the reader.

Values	Principles	Primary Practices	Corollary Practices
Communication	Humanity	Sit Together	Real Customer Involvement
Simplicity	Economics	Whole Team	Incremental Deployment
Feedback	Mutual Benefit	Informative Workspace	Team Continuity
Courage	Self-Similarity	Energized Work	Shrinking Teams
Respect	Diversity	Pair Programming	Root-Cause Analysis
	Reflection	Stories	Shared Code
	Flow	Weekly Cycle	Code and Tests
	Opportunity	Quarterly Cycle	Single Code Base
	Redundancy	Slack	Negotiated Scope Contract
	Baby Steps	Ten-Minute Build	Pay-per-Use
	Accepted Responsibility	Continuous Integration	
		Test First Programming	
		Incremental design	

Table 1: Values, Principles, and Practices of Exterme Programming

Feedback is discussed as a value within XP to cope with the unknown and/or evolving expectations of what is to be delivered and the coding techniques for delivering it [Figure 14]. XP values short and rapid feedback loops, even as fast as multiple times per day. As discussed by Leffingwell above, Beck asserts that requirements and plans built ahead of time become irrelevant quickly, thus information feedback from what has been developed to date is needed to keep the plans relevant. The communication value supports this need for feedback by identifying the importance of distributing the information.

Several principles exist to realize value from the feedback. Flow is a principle that calls for the steady delivery of software products by identifying items of incremental value, completing all of the design, test and delivery activities in quick succession so that the results can provide value to the customer. It also provides information back to the organization so that it can learn and adjust right away. Reflection is a principle that calls for the regular consumption of the feedback information and then to act on it. Reflection occurs from the perspective of identifying improvements to delivery methods and from the perspective of adjusting what is being asked for based upon the learning from what was just delivered. The practice of identifying "Stories", small pieces of functionality that can be developed and delivered in a week, in conjunction with the practices of the ten-minute build, weekly and quarterly cycles, incremental deployment, and continuous integration enable the team, the larger organization, and the customer to gather information regarding product attributes and the capabilities of the development organization.

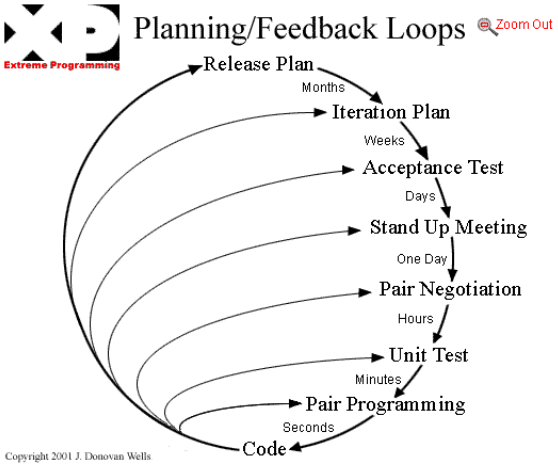


Figure 14: Extreme Programming Feedback Loops [20]

Economics is also called out as an important principle. The discussion about economics cites the importance of the time value of money. Deploying and selling products sooner is valuable to the business, and the practices identified above related to providing rapid feedback also enable the scope that is complete to be packaged and sold right away. In addition, Beck promotes the practice of negotiating pay-per-use or subscription payment models allow the financial structure to decouple business equation from the fixed scope model. It allows updates to flow to existing consumers more readily and enables businesses to generate revenue in the form of new subscribers by allowing new capabilities to be released sooner without waiting for larger, less frequent deployments of pre-defined scope.

What is notably missing from Beck's book is an explicit framework. Beck explains that his definition of XP has evolved beyond that of a methodology to "…a style of software development focusing on excellent application of programming techniques, clear communication, and teamwork…". His colleague and collaborator on XP, Don Wells, has published his own description of XP [20] along with diagrams which summarizes the flow of 'stories' from creation to delivery [Figure 15]. From these diagrams, we can better understand how project management and prioritization come into play. Stories are small units of deliverable scope that have sufficient description to allow a developer to act on them. In XP, stories are no larger than what can be completed in a week, including design, coding, building, and testing.

Customers are responsible for determining the stories and the priority of the stories. The team helps the customer create the stories by completing small research activities, called “Spikes”, that provide information to refine the stories. Stories are aggregated into a release plan, which is the list of stories desired for the next release. This list feeds the iteration plan, which is the list of things that the developers will do in the next iteration. Iterations are a fixed period of calendar time that the team plans to deliver solution to the stories agreed to as part of the iteration plan.

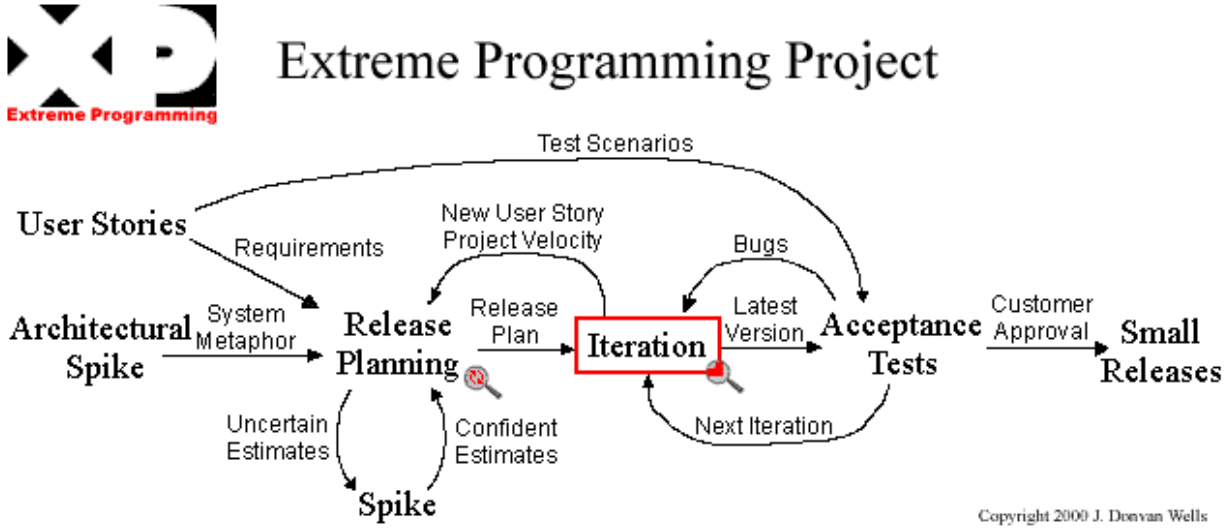


Figure 15: Extreme Programming Framework [20]

2.9 Summary: Project Management within Classic Agile Methodologies

The Agile Manifesto, Agile principles, and three classic Agile frameworks, as expressed by their authors, have been developed with the primary goal to care for, and enable software development teams to deliver efficient, and high quality software, and to allow development teams and organizations to continuously improve upon their practices based upon feedback information provided by the frameworks. The fundamental organizational unit that the frameworks are built around is the single software development team. The team is self-organizing, long lasting, and is assumed to have all the needed design, build, and test capabilities present within the team and that anyone on the team can accomplish any of these duties. The authors of the frameworks do speak to scaling of the frameworks to larger organizations, but these techniques are not formalized within the frameworks.

Agile frameworks are built on the assumption that the inbound work requests fit into short delivery increments, ranging from one week to one month in size. Completion of all design, build, and test activities for the work request is expected to be completed within the delivery increment. This enables the strategy of building frequently and then reacting to what is learned from using/testing the build product. Creation of these small units of requested scope is the responsibility of the customer, or the agent of the customer. For each of these requests, the customer also defines what it takes for the requested work to be considered done. The developers on the team have the right to refuse to work on

the requested work item if it is not defined clearly enough or if it is too big. The customer is expected to be embedded within the team so that active negotiation and collaboration can happen as needed.

A team that has embraced Agile will not take on project management ownership by any of the developers. For each of these frameworks, the list of requested units of scope are created by the customer (or agent) and placed in prioritized order. The development team simply pulls in the next request when it has capacity to do so. In effect, prioritization decisions for project scope within the team do not exist.

None of the frameworks address the prioritization of administrative or teaching/learning activities at all, but Beck writes about how the small size of the units of requested scope naturally addresses the need for collaboration outside of the team because the scope of the request is so small that there is no need for coordination. This implies that the customer (or agent) has taken the responsibility to resolve any coordination needs while making the prioritized list of requests. A second trait of these frameworks that removes dependencies is that all developers have equal access and edit rights to all parts of the code for the system, thus removing any dependencies related to access.

2.10 Scaled Agile Framework (SAFe)

Scaled Agile Framework (SAFe) [21] [22] is the decades-long culmination of the work of Dean Leffingwell which provides guidance and a framework to address the key the challenges when trying to scale up the classic Agile frameworks discussed above:

- Operationalizing the classic Agile frameworks across large organizations with hundreds or thousands of software developers.
- Providing methods and structure required to provide the small units of work assignments that allow development teams deliver incrementally using Agile techniques.
- Providing frameworks to build project and portfolio planning and management activities that are tailored to the attributes of products that are built from software.
- Providing a roadmap and guidance to transform entire enterprises to SAFe

The SAFe framework is summarized in Figure 16. In addition to being the title of the framework, SAFe is the business brand name for the large consulting, training, and certification business that he has built around his work. What follows is a discussion of several aspects of the framework that pertain to the research at hand.

At the team level, SAFe leverages a mixture of the classic Agile frameworks, utilizing the repeating cadence and roles of Scrum to provide the initial structure. XP practices are called for to enable incremental deliveries and product quality. Kanban techniques at the team level provide feedback measurements of throughput and help to identify bottlenecks and process improvement opportunities. SAFe does speak to the importance of continuous improvement and identifying impediments that slow down the delivery of the team, but these aspects are discussed proportionately less than the classic Agile frameworks, in favor of the topic of project management.

The SAFe framework places heavy emphasis on providing for alignment across teams. Several layers of project management are provided within the framework. These layers have responsibility to process and prioritize customer requests into a coordinated set of action items for the development teams to pull into their work queues. SAFe synchronizes the calendar of all the design teams so that they start and end their design iterations at the same time. This aligns the planning times, which enables teams to coordinate their activities and manage interactions by actively planning time into their schedule for the work.

Coordinated team planning is further enhanced by defining a planning time unit larger than a sprint, called the Program Increment (PI), which aligns with a fixed number of sprints. The planning increment serves to identify a larger unit of planning and delivery cadence. Coordinated Program Increment Planning Events, bring design teams together, in the same room if possible, to plan the next larger increment of value that will be delivered. In the planning event, everyone first hears from the product managers about the vision and bigger picture goals for the next Program Increment. Then all the teams, with heavy support of their product owners pull work requests into their team queues and define their incremental deliverables for each of the design iterations of the next program increment. This event provides for alignment of work across many teams to converge to the delivery of larger, more complex systems.

The Program Increment Planning Event is the central activity where project planning efforts are brought to the scrum teams and where the management of dependencies across teams occurs. When the teams finish their planning work, they present what they intend to deliver in the next program increment.

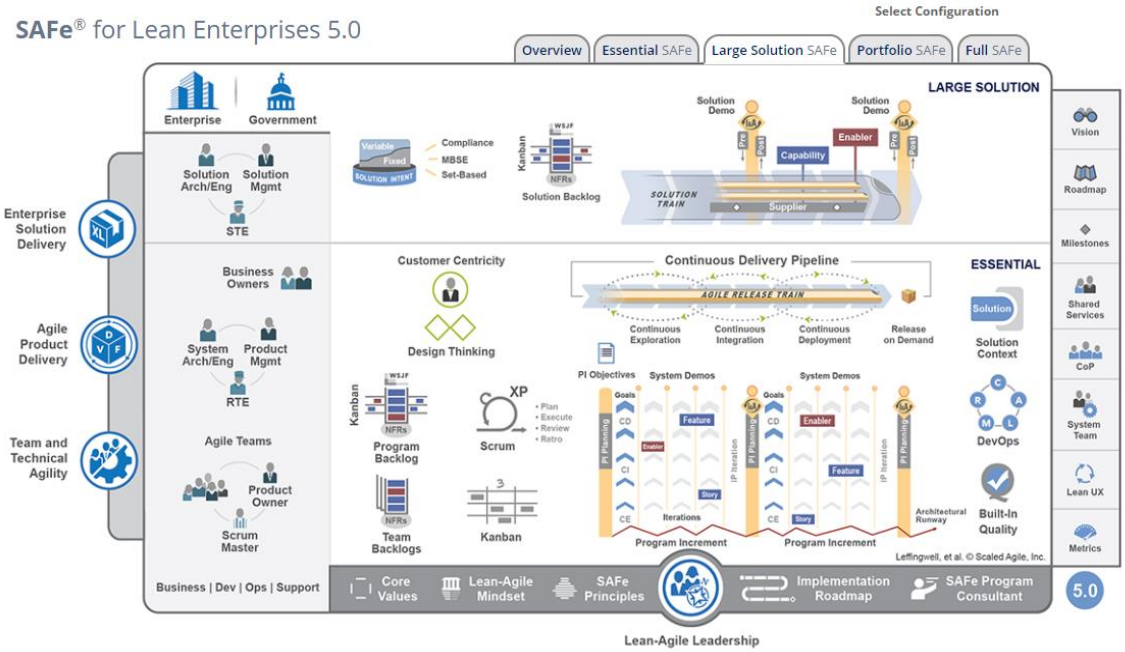


Figure 16: Scaled Agile Framework (SAFe) [22]

Product managers can then see immediately if the aggregate plans across the teams are aligned and are likely to achieve the goal. If needed, adjustments to the team plans are made on the spot.

The SAFe framework addresses the long-term planning gap of classic Agile frameworks by defining roles and methods for processing the requested scope into units small enough for the design teams to act upon. SAFe defines several project management roles in a hierarchical manner. Each tier has larger swaths of ownership of the overall scope that the enterprise has in its portfolio. At each level, a backlog is created to identify the projects to be completed and to manage the priority. Each item in the backlog at a higher level is decomposed into multiple smaller items for the backlog(s) at lower levels. All levels of the hierarchy work together to first understand what is being asked for and then decompose these requests into the units of scope that product owners and teams can pull into their team backlog. Once pulled into the backlog, they further decompose the scope into units that they can deliver in a design iteration.

An output of the development team's planning activities is data regarding how long it took to complete the backlog items. This data, if kept over a long time, becomes the basis for project estimation to occur. Since the backlog items are derived from larger items on the next higher backlog, the data can be applied to the next higher backlog items to provide an estimation.

In addressing the planning gap left by classic Agile methods, the SAFe framework re-introduces the elements from systems engineering and systems architecture disciplines found in traditional Stage-Gate planning activities. SAFe uses the term "Solution Intent" to represent stakeholder wants and needs, ConOps information, and requirements capture. It has multiple System Architect roles. SAFe also calls out the value of Model-Based Systems Engineering (MBSE) as a best practice. SAFe uses these roles and activities to support the design of the teams' stories so that developers complete the stories with a minimum of external dependencies, and an improved likelihood that the stories defined will aggregate to the desired solution. The framework emphasizes that, while these activities and roles are working on this up-front and ahead of the design teams, it is still not like a Stage-Gate approach because planners complete the up-front work only to the fidelity required for the current needs. For example, the product architecture may be designed at a high level to identify the product modules' primary interfaces and work to specify only one of the subsections in detail. Then the architects work on the details for the second module while the software developers work on the first module. This allows learning from the first module's delivery to influence the design of the subsequent modules and supports a continuous flow for the architects and systems engineering teams. The flow-based structure keeps an organization and process in place that can react to the continuous scope discovery and new requests from stakeholders.

Literature for the SAFe framework also includes a framework for deploying SAFe into an enterprise. This deployment framework emphasizes the need to start at the top with an enterprise design activity, which first analyzes the customer’s value stream that the organization is providing software for. This is known as the Operational Value Stream. The software systems that the operational value stream needs to complete the mission are identified. Then the value streams that provide software development activities required to deliver the software are designed by the enterprise. These are known as the Development Value Streams. See Figure 17. Kanban charts are used to capture this design and development teams are organized around these delivery value streams. This up-front analysis allows for process optimization to occur, and it also allows for aggregation of the right skillsets. Large projects will have multiple value streams that deliver different parts of the product needs for the customer. Teams working within a given value stream are aggregated together for the Program Increment planning

events. These groupings are called Agile Release Trains. This pattern allows for the SAFe framework to scale to any size organization.

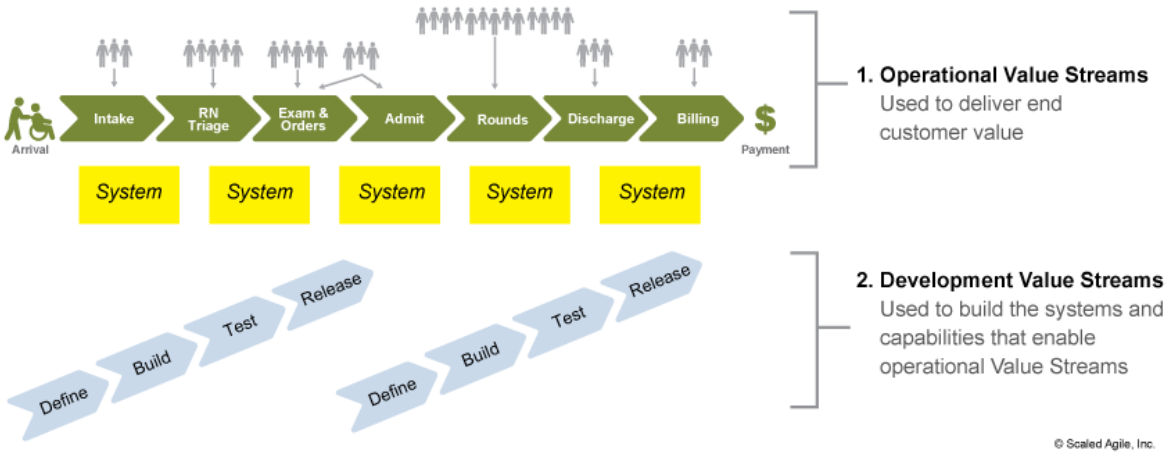


Figure 17: Development Value Stream Identification [22]

2.11 Large Scale Scrum (LeSS)

Large Scale Scrum (LeSS) [23] is another example of a scaled agile framework intended to work on larger organizations and larger projects. This framework builds upon the Scrum framework discussed above but has some adjustments to the product owner role and backlog management. First, the framework calls for the teams to work on a single project with a single Product Owner. There is not an accommodation for multiple projects running through the same set of development teams. The Product Owner is responsible for the product backlog that is being built by the development teams. If the project has more than four to eight development teams working on the product, the framework inserts additional layers of "Area Product Owner" roles to manage the large number of teams. Area product owners decompose the larger project into the small units of scope, known as"Stories", required for efficient delivery team execution. It strives to keep the organization very lean. The framework calls for the addition of an area product owner for every four to eight scrum teams working on the same project. See Figure 18 [24] for an illustration of the product owner hierarchy.

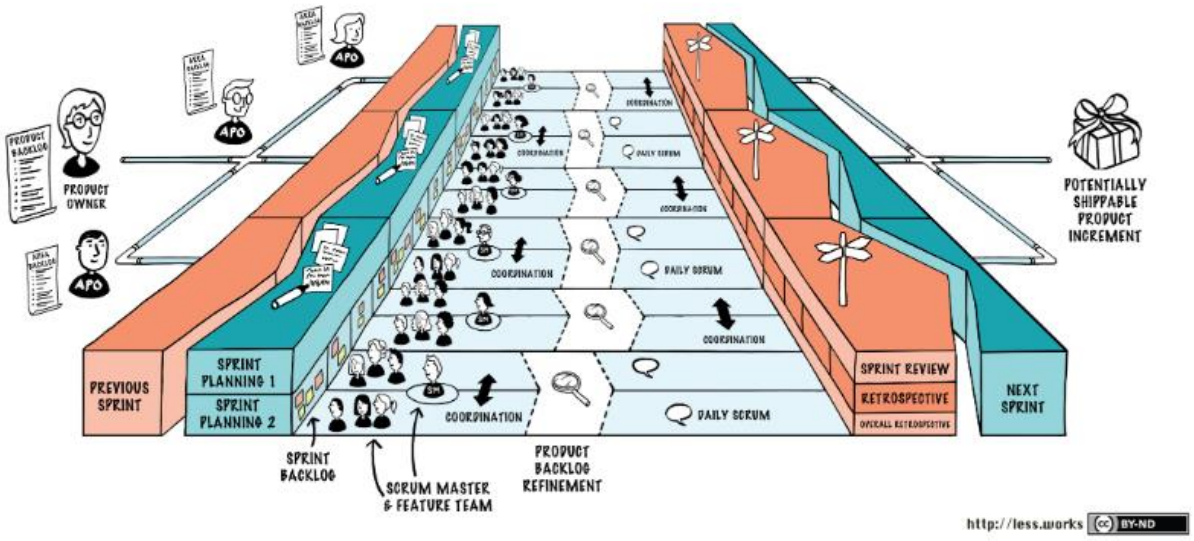


Figure 18: Large Scale Scrum Framework for Large Numbers of Teams [24]

Coordination across the many teams occurs through three formalized methods. The first method is through the area product owners' efforts as they create the product backlogs for the different teams. The second is through an additional sprint planning event where all the teams in a given area meet together with the area product owner and coordinate the necessary interactions. This extra sprint planning happens on the same cadence as the regular sprint planning. Finally, the "Overall Retrospective" occurs, where the lessons learned are shared, and requests for process improvement are coordinated and aggregated into a unified request.

Unlike the SAFe, there is no larger planning horizon provided by Large Scale Scrum. Large Scale Scrum does not speak about requirements or systems engineering activities. It does not provide for portfolio planning at an enterprise level. LeSS does speak very directly about the framework's goal to expose issues within the project delivery system. It defines the role of managers and leadership as being responsible for improving the project delivery system. They take care of the people and work to remove impediments that slow down the teams' work. In support of these efforts, the LeSS framework's principles emphasize systems thinking methods to improve the system that exists for delivering the product. These principles include systems thinking, Lean thinking, empirical process control, and queuing theory.

2.12 Hybrid Approaches

Another concept that has emerged for managing projects with large amounts of software content is known as "Agile-Waterfall Hybrids." As the name suggests, these methods blend the principles and methods of Agile and Stage-Gate principles and methods.

Cooper [25] offers the following table to describe the differences between Agile and Stage-Gate methods. He describes Stage-Gate frameworks as "Macroplanning" strategies that include all the activities related to the conceptualization of a product, all the way through the manufacturing, support, and retirement. It includes all the functional disciplines required to proceed through the project lifecycle, including manufacturing, engineering, marketing, and product service.

Table 2: Differences between Agile and Stage-Gate Management Practices [2]

	Stage-Gate	Agile
Type	Macroplanning	Microplanning, project management
Scope	Idea to launch	Development and testing, can be expanded to pre-development
Organization	Cross-functional team (R&D, marketing, sales, operations)	Technical team (software developers, engineers)
Decision model	Investment model—go/kill decisions involve a senior governance group	Tactical model—decisions about actions for next sprint made largely by self-managed team

Agile methods are more tactical in their planning horizon when compared to Stage-Gate. They focus specifically on the planning of engineering activities completed by the engineering and testing teams. They focus on execution and provide discipline and structure that allow engineers to stay focused on the task at hand. They also focus on methods to provide feedback as working solutions that can be evaluated to validate that the planned scope aligns with the customer's value.

Cooper's research indicates that the typical hybrid system applies agile techniques to the design and test stages of the Stage-Gate framework to leverage the benefits of Agile methods. He cites research indicating positive results of such a hybrid.

Another example, as illustrated by Bergman and Hamilton [26], speaks of hybrid systems that have Agile methods applied to the software's design while the mechanical systems are developed using Stage-Gate methods. This allowed for each solution medium to utilize the method best suited to it.

In these examples, the authors note that merging methods are not without effort and change to meet both sides' expectations. They also cite the need for macro planning and leadership's willingness to change how they monitor and lead projects. The fidelity of the tactical planning that Agile brings allows engineering teams to plan to full capacity, thus forcing leadership to deal with the difficult prioritization decisions instead of relying on engineering teams to absorb new requests into undocumented slack. Secondly, the feedback information from Agile teams does not readily translate into standard stage gate monitoring reports. This means leadership must have it translated, or they must learn the new reporting formats and what the data tells them about overall progress and performance.

2.13 Velocity and Burn-Up Charts

The concept of "Velocity" is leveraged within SAFe and used to estimate future progress with the backlog delivery. Velocity is defined as the amount of work a team can complete in a given amount of time, such as a sprint or a delivery increment. The amount of work done is typically in the units of a point system used to estimate the size of the work items in the backlog.

In practice, a team identifies its velocity by first establishing a routine of estimating the effort in "points" that each of the backlog items will take. This activity can be known as "sizing" the backlog. The definition of a point is relative to the team, although SAFe calls for the units to be somewhat consistent across the teams, such as using a single day of work as a "point." The team then tracks how many points it can complete in each delivery increment, and the results are averaged over several increments to determine the "Average Velocity." A team can use its velocity and sized backlog to estimate how long before it will deliver a certain amount of desired content. The team calculates this by dividing the total points included in the milestone by the team(s).

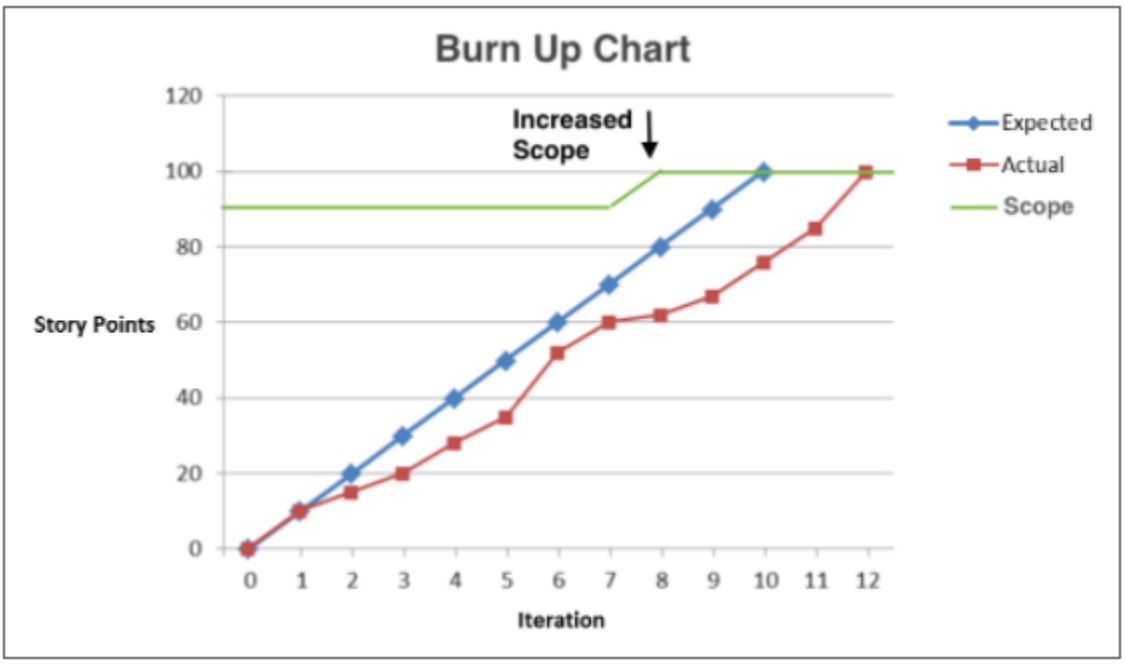


Figure 19: Example Burn-Up Chart [27]

Leadership monitors ongoing progress by comparing the actual progress by the expected progress at any given point in time. It can be summarized and presented using conventional burn-down charts common to the field of project management. However, burn-down charts miss a crucial item that Stage-Gate frameworks ignore, and Agile embraces: scope is subject to change as developers complete work. Therefore, the Burn-Up chart is used. Figure 19 shows a representative burn-up chart, as summarized by Vergini [27]. The chart tracks two items. One is the work, in story points, that the team(s) complete, and the other is the growth of the scope included in the milestone. The increase in scope can come from improved estimates, discovered scope, or new requests added to the milestone. The main advantage of expressing the data in this way is that it provides more insight into the reasons why a milestone is not being reached.

2.14 Summary of Project Management Techniques

Table 3: Summary of Project Management Tools per Framework

Framework	Actual Performance Data	Expected Performance data	Communication Tools	Agent to the Customer	Interface with the requestor
Stage Gate	Time spent, Money spent, Tasks completed,	Work breakdown structures and up-front estimates on tasks, money, and time required,	Work breakdown structures, Gantt charts, critical path charts, earned value charts	The project management organization led by the project manager	Gate reviews, Progress meetings
Scrum	Story Points completed (Velocity)	Sprint commitments	Backlogs,	The product Owner	Sprint Demos, Backlog Grooming
Kanban	Story execution consistency	Stories per Sprint	Backlogs, Kanban charts	The Product Owner	Sprint Demos, Backlog Grooming
XP	Stories completed per sprint	None specified	None specified	The Product Owner	Sprint Demos, Backlog Grooming
SAFe	Story Point estimates for stories, features, and epics	Story point estimates for, stories, features and epics	Velocity, Burn Up Charts, "Sized" Feature and Epic backlogs	Product Owners, Project Manager, Solution Manager,	Program Increment Demos, Planning, and Retrospectives
Large Scale Scrum	Story point estimates	None specified	Velocity calculations, Burn up charts	The product owner	Sprint Demos, Backlog Grooming
Hybrid Waterfall/Agile	Not stated	Not stated	Not stated	Product Owners and Project managers	Not stated

This section extracts the data, methods, and agents involved in answering this question from the reviewed literature. Table 3 summarizes the roles and tools provided by each framework to forecast the scope's delivery date. Each row represents one of the reviewed frameworks. Column one contains the name of the framework. Column two lists the data that could be used as part of the determination of the teams' actual performance. Column three identifies the activities used for forecasting scope at a given date. Column four identifies the instruments or tools used for communicating the information. Column five identifies the key people involved with the creation, processing, communication of the forecasts. Column six lists any formalized interface structure or activity used to share the forecast data.

As discussed earlier, Stage-Gate frameworks assume that the scope can be understood entirely up-front, and the planning and forecasting activities leverage this assumption by first completing a work

breakdown structure for each of the stages. Then estimates of the time and cost for completing each activity are generated. The results are captured using a Gantt chart or some similar format that can summarize the estimation results. Progress is monitored by comparing the actual time or money spent with the original plan. Communication occurs through formalized gate reviews and periodic meetings of project management teams, including the project manager and representation from the design groups involved.

The classic agile frameworks, which focus more on managing the flow of tasks through the team's design activities, provide backlogs centered on each team as the primary means of identifying the work to be done. The product owner is designated as responsible for creating the backlog, but backlogs are living documents that are constantly revised to account for scope discovery, learning, and new requests. There is no specific guidance for creating forecasts beyond the current sprint. All three of these frameworks identify a product owner as the agent for creating and prioritizing the backlog and representing the customer. Ideally, the frameworks call out that the customer should act as the product owner, but there is an acknowledgment that this is not always possible. Planning sessions and demonstrations are the means for communicating actual progress outward from the team. It seems implied that this communication will allow stakeholders to formulate any forecasts that they deem necessary as an activity outside of the framework.

The three classic Agile frameworks vary somewhat with the generation of forecast data. XP does not call out a specific method, relying on the prioritization activity alone. The argument for this is that if the design team is working on the most valuable thing and can release, longer-term estimations are not needed. The Kanban framework relies on the team and product owner arriving at the creation and delivery of stories sized to fit within the delivery increment. Then the stories per increment data can be applied to the backlog to create a forecast. The Scrum Guide [16] does not directly speak of methods to forecast beyond the current sprint commitments. All the classic Agile frameworks leave predicting the completion dates of future work unaddressed.

Large Scale Scrum includes more agents and effort within their frameworks for planning and decomposing larger projects into smaller units. Large Scale Scrum identifies more product owner roles to manage the distribution and governance of priorities of stories across larger numbers of teams. However, it does not add to the technique of using story point estimation for projecting on the expected progress.

The SAFe adds even more structure, roles, and effort to the planning activities. It defines and utilizes the concept of "Story Points" and "Sizing" activities to create delivery forecasts for items in the backlog. It also introduces a multi-tiered backlog approach that allows for larger, lower fidelity activities called "Epics" to be created and estimated early on. These activities are then decomposed to a middle-sized "Feature," which can be sized before being decomposed again to the "Story" that teams will complete. This framework provides a method for estimations to be created that reach further into the future with larger, more abstract, work units listed. This can provide better long-term planning for large projects.

The literature reviewed lacks detail regarding how hybrid Agile Stage-Gate systems provide for the activities that define the expectations gap. All the documents reviewed speak of the continued existence of product owners and team backlogs. The documents also identify that Stage-Gate activities exist for the non-software part of the project system. The documents offer no detail regarding what the

management team for the overall project is using for data or communication tools to formulate the Agile teams' expectations gap. This study intends to expose the details related to how this works.

3 A Model for Analyzing Project Organizations

This chapter shares the method used to analyze the project delivery systems' structural and procedural aspects under study. It begins with an introduction to causal diagrams and the application of systems dynamics modeling to projects. It continues with a discussion of what a causal analysis reveals when using the technique to describe the delivery teams' and supporting organizations' actions and behaviors that exist to deliver sequential and concurrent software projects.

3.1 Causal Diagrams and System Dynamics Modeling

As discussed above, our system of interest is the project delivery system, a sociotechnical system responsible for delivering solutions for mechatronic products. This system includes the software developers and the agents and teams that lead and support the development teams. As such, the dynamics of this system are of particular interest because the effectiveness of the mechanisms that excite or control the dynamic interactions between the agents, activities, structures, and behaviors will impact the organization's emergent performance.

Causal diagrams are a useful tool to express the nature of the relationships between attributes of the system [7] [6]. Causal diagrams are expressed as sequences of nodes and edges. Nodes represent the variables of interest within the system, and the edges represent the links between them. Edges are directional and are also assigned a positive or negative sign, representing the type of relationship. Positive signs identify that an increasing value of one node increases the value of the other node. Negative signs identify that an increasing value of one node decreases the value of the other node. The direction of the arrow represents the direction of the cause and effect relationship.

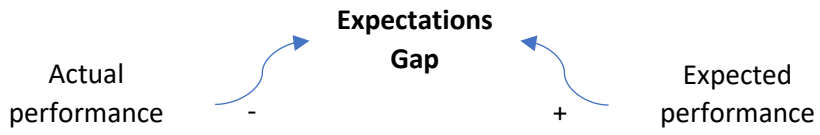


Figure 20: Fundamental Causal Relationship of an Expectations Gap

Figure 20 illustrates this study's relationship of interest in a causal diagram format. The diagram represents perceived performance in terms of an "Expectations Gap" that results from the difference between Leadership's expected performance and the team's actual performance. This diagram indicates that the expectations gap increases with a decrease in actual performance or an increase in the expected performance.

Lyneis and Ford [6] discuss the use of causal diagrams to frame a model that describes how governing actions reverberate through the sociotechnical system that is delivering a project. The causal diagram is merged with the stocks and flows modeling concept, Figure 21, that expresses scope flowing from a stock of “Work to Do” to a stock of “Work Done” or through a rework loop.

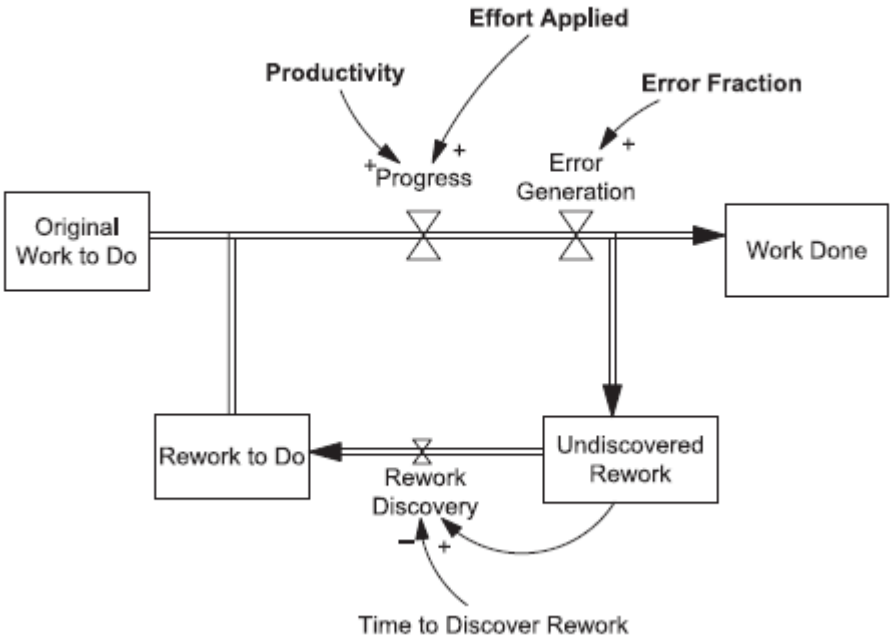


Figure 21: Basic Stocks and Flows Model for Project Systems [6]

Januszek proposed an expansion to the basic stocks and flows model to account for Agile methods, as summarized in Figure 22. His model accounts explicitly for the discovery of scope. At the same time, the model includes stocks of "Unknown Unknowns," which is the scope needed to deliver upon the project's expectations but is not identified at the beginning of the project. Unknown Unknowns are discovered during regular development activities or are identified as risks which will require explicit scope to be added to mitigate the uncertainty related to them. This model subjects the discovery of Unknown unknowns to an approval process while the risk mitigation is added directly to the product backlog.

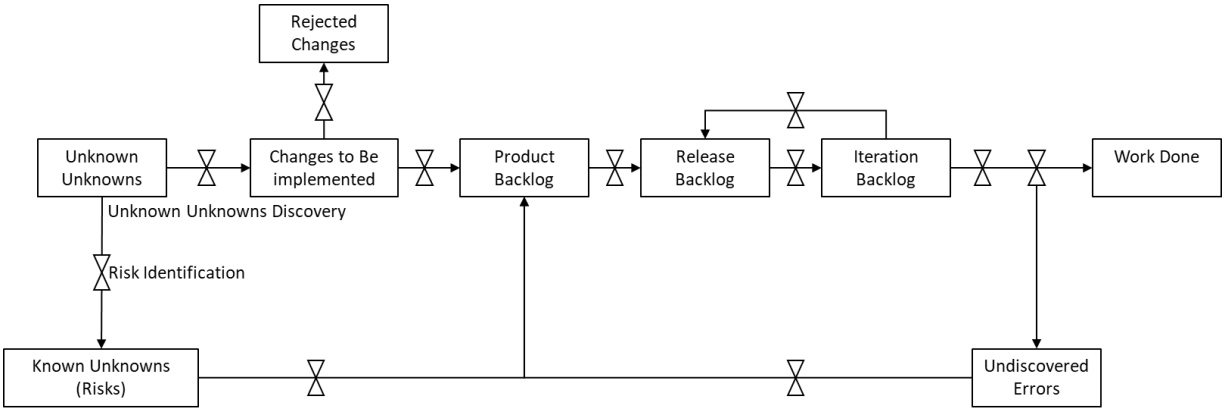


Figure 22: Agile Stocks and Flows Model per Januszek [7]

Combining the causal diagram concept of Figure 20 with the stocks and flows diagram of Figure 21 provides the causal relationships between the actions taken to control the system's stocks and flows and how it ultimately impacts the delivery of scope. Figure 23 is an example of such a combination. The application of system dynamics for modeling project behavior has been considered a successful endeavor with many extensions of the fundamental model utilized for many project scenarios.

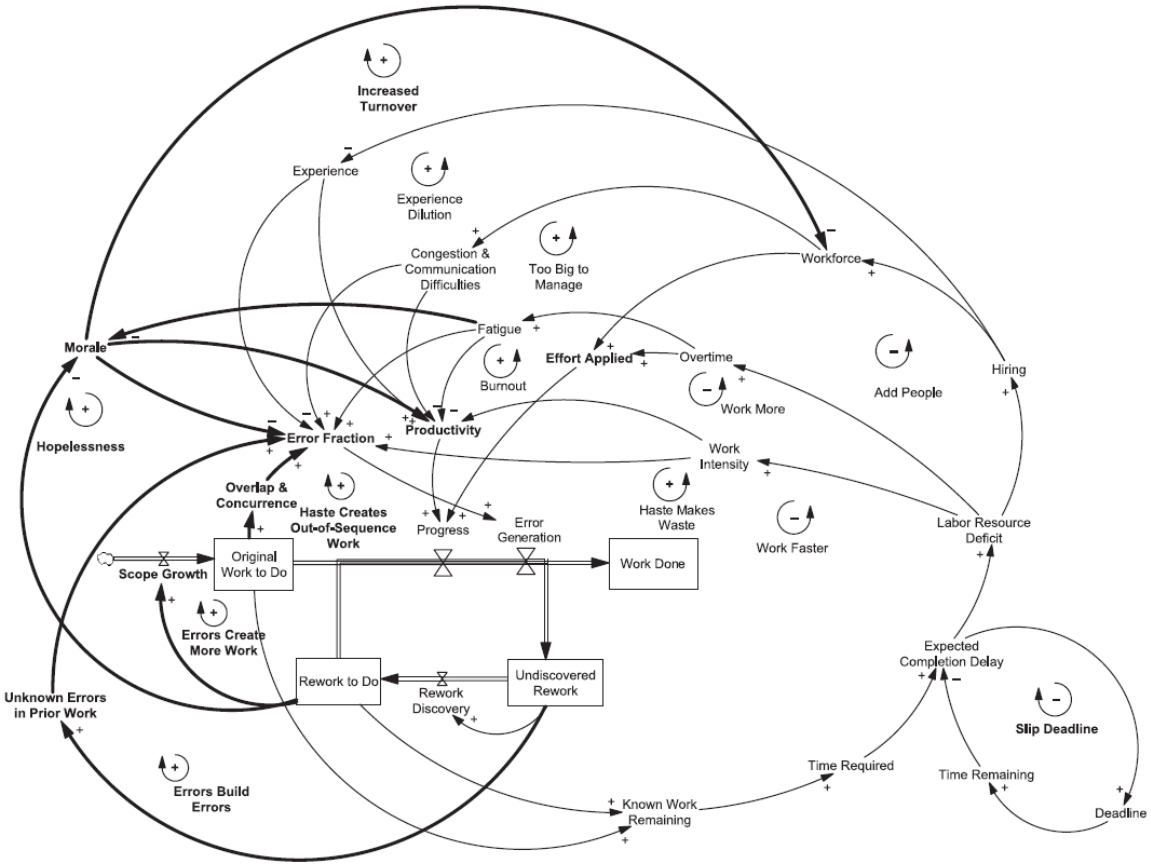


Figure 23: Systems Dynamics Model of a Typical Project System [5]

While the actual creation of an executable model that can simulate a project's dynamics is not the intention of this thesis, causal diagrams are a powerful modeling tool that will be leveraged for this analysis.

3.2 A Model for Designing and Managing the Dynamics of a Project Delivery System

Based on the research summarized above, the following framework is proposed, allowing one to appreciate the dynamic aspects of a project delivery system. This model is intended to represent the dynamics in the project delivery system, regardless of the management methods applied within the system. This model is created to assist with expressing and processing the research activities of this study. However, it may also be useful as a general framework to assist in the design, analysis, or active management of new or existing project delivery systems.

This model's perspective is defined to be a project delivery system that consists of the organization, tools, processes, people, and skillsets engaged in delivering software solutions for a single project, or many ongoing projects running through the project delivery system at the same time. This is an expansion of the models of Lenius and Ford, and Januszek, presented above, in that their models focused on a single project. Figure 24 illustrates the core of the model, where the software development

teams, residing within the Team System, hold a central role. These teams deliver product scope to the larger mechatronic project. Their scope delivery activities can be measured in terms of speed, quality, and cost and are represented as Actual Performance in the model. This Actual Performance is compared to the Expected Performance, the performance levels that the Project Leadership has for the software development teams. The Expectations Gap is the difference between what leadership expects from the software development teams and what is delivered.

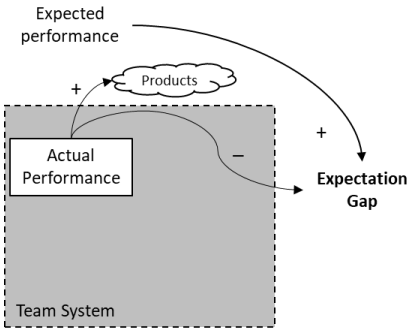


Figure 24: Definition of the Expectations Gap within the Project Delivery System

This thesis will focus on the causal loops that affect how the Actual Performance and the Expected Performance change due to control activities and project management mechanisms that evolve within the organization. For that reason, the stocks and flows portion of the classic systems dynamics model occurs within the Actual Performance node, where the level of actual performance represents the success with converting "Work to Do" to "Work Done."

The Expectation Gap node represents the project leadership's perspective regarding how well the Agile development teams are delivering against what the project leadership is expecting. This may be in terms of cost, schedule, scope, quality, or other factors, but we will concentrate upon the delivery of the desired scope on a regular schedule for this study.

The Expected Performance node represents what project leadership expects to have delivered by the software development teams. This is in terms of the amount of delivered product at the expected schedule, quality, or cost. It is an input to determining the Expectation Gap and is the central area of emphasis for this study. In this study, project leadership is working in a Stage-Gate framework above the software development teams that govern themselves using Agile methods.

The Actual Performance node represents the work completed by software development teams. Of interest in this study is the metrics related to describing the performance of the team. For example, in a Scaled Agile framework, the performance metrics would include measuring the delivered scope in story points and perhaps a historical record of story point deliveries per sprint.

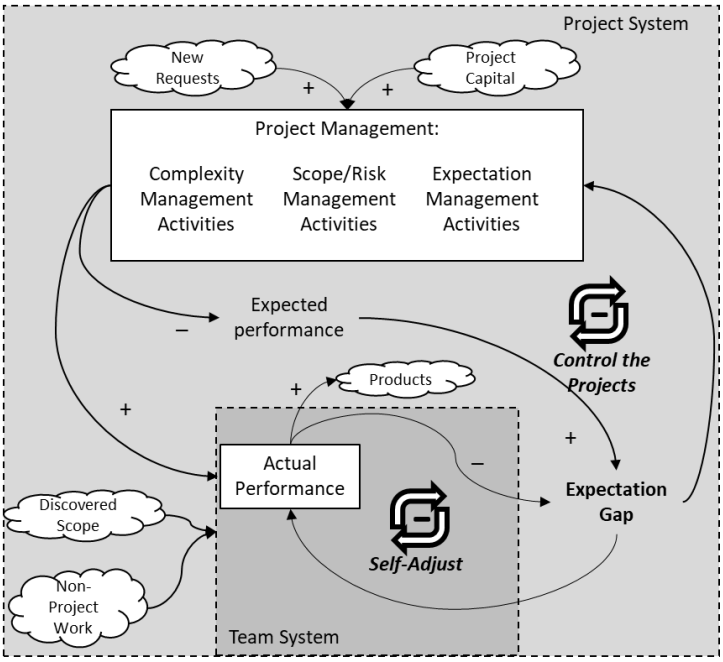


Figure 25: Project System Level of the Project Delivery System Model

Figure 25 expands on the Team System by introducing the project management activities. It also illustrates two control loops that can exist to govern the actual performance. The Self-Adjust loop represents the actions that the development teams take on that are within their control to improve their performance. The Control the Projects loop represents the project management activities that a governing team takes that change the level of Actual Performance or the Expected Performance to reduce the Expectations Gap.

There are three sources of scope for the Project Delivery System Model. The New Requests node, represented as a cloud in this diagram, is one source that drives activity within our Project Delivery System. This model applies to systems that are executing more than one project within the system; therefore, the new requests can be for any number of projects that the Project Delivery System is delivering upon.

The Discovered Scope node represents additional work requiring completion to deliver to the goals/requirements of the pursued project(s) but not identified up-front. Therefore, it is not factored into the original Expected Performance value that the leadership has for the project(s). It also includes dependency management and coordination activities, as discussed within the literature review, which can be considerable with developing complex products. The Discovered Scope is represented as a cloud,

which implies an unlimited stock exists, but this is a simplification for graphical purposes. The reality of discovered scope is that it is bounded by the project's defined expectations, as Januszek demonstrates within his model.

The Non-Project Work node represents those requests upon the development teams that is not part of any official projects. It may include administrative activities, training, support or troubleshooting of completed products, or work on unofficial "side projects" that the team gets recruited to support outside of the defined Project Management activities.

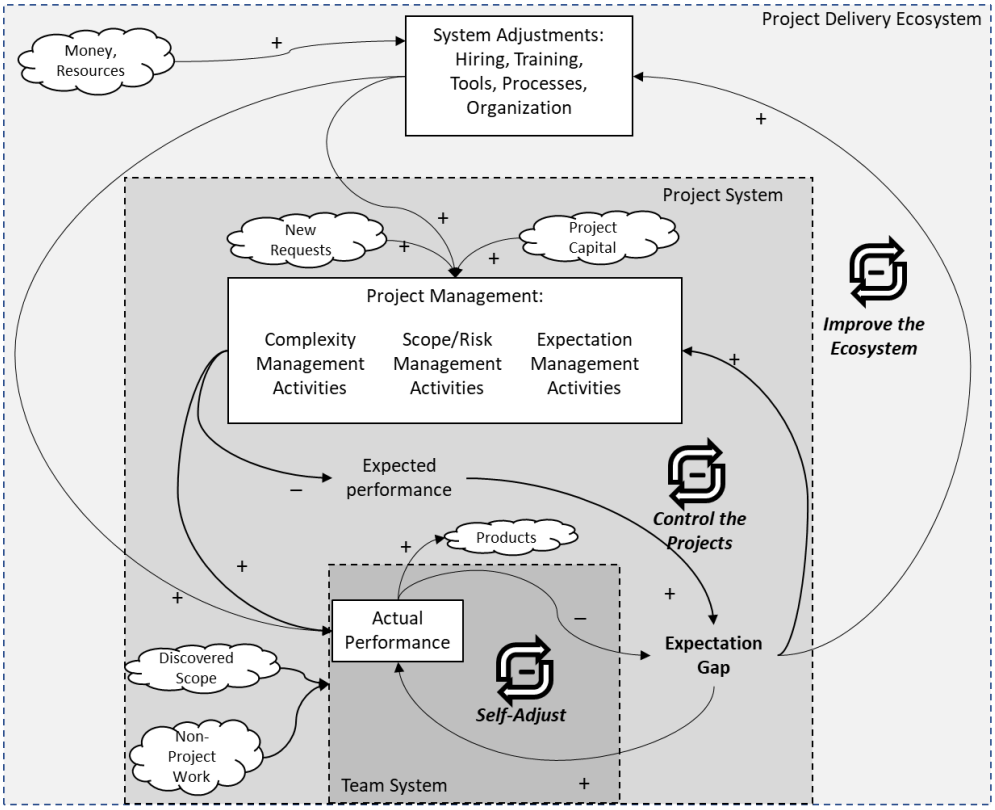


Figure 26: Ecosystem Management Loop of the Project Delivery System Model

The Project Management node represents the completed activities that define and control the project(s) as it progresses towards its goal. This node delivers guidance, priorities, work breakdown structures, architectures, requirements, and other information to the development teams and project leadership. The activities completed within this node influence and define the Expected Performance that project leadership has for the software development teams. These activities may occur within different parts of the organization, including the development teams, project management teams, and systems engineering and architecture teams. The agents of this node include the program/project management team for the overall mechatronic system and the product owners of the software development teams. The Project Management node has three categories of activities within this node, Complexity Management, Scope/Risk Management, and Expectations Management. These sub-activities are further described later in this section. As illustrated in Figure 25, the Expected Performance is derived from project management activities. It may be dynamically adjusted over time as new information is available from the Project Delivery System's overall development activities. Agile methods and Stage-Gate

methods differ in that Stage-Gate methods generally expect changes in expected performance to be an exception. Therefore, the need for constant adjustment may not be provided for, while Agile methods only provide monitoring Actual Performance and do not cover the need to change the Expected Performance.

Figure 26 illustrates the Ecosystem Layer of the project delivery system model. This level represents the overall structural, organizational, and procedural attributes of the system and the management activities used to adjust them. While the inner layers focus more on project execution, this layer controls and enhances the project delivery system's performance.

The model nodes account for all the activities of project management and execution. PMBOK [2] provides the project management activities considered to be best practices for an effective project. Table 4 summarizes the relationship between the project management activities discussed in PMBOK and the project delivery system model's nodes.

Table 4: Project Management Activities for Each Node of the Project Delivery System Model

Complexity Management	**Scope Management**	**Expectations Management**	**Leadership Adjustments**	**Actual Performance**
Collect Requirements	Develop Charter	Estimate Activity Duration	Project Management Plan	Deliver Scope
Create WBS	Direct and Manage Work	Manage communications	Plan quality management	Control Quality
Identify Risks	Monitor and Control Project Work	Monitor Communications	Plan Resource Management	Manage Quality
Perform Risk Analysis	Perform Integrated Change Control	Monitor Stakeholder Engagement	Acquire Resources	Implement Risk Responses
Define Activities	Close Process Phase	Manage Stakeholder Engagement	Manage Team	
Sequence activities	Define Scope	Schedule Management (Identification)	Control Resources	
Plan Risk Responses	Validate Scope	Develop Schedule	Plan Communications Management	
	Control Scope		Plan Risk Management	
	Estimate Costs		Implement Risk Responses	
	Determine costs		Plan Procurement Management	
	Estimate Resources		Control Procurement	
	Plan Risk Responses		Conduct procurements	
	Monitor Risks		Plan scope management	
	Identify Stakeholders			

Like all systems, a project delivery system exists within still larger systems. Figure 27 illustrates how the Project Delivery System model can scale recursively to handle more complex project delivery systems. The situation under study could fit into this scenario in that there may be a complete software project delivery ecosystem that resides within a broader ecosystem for mechatronic projects.

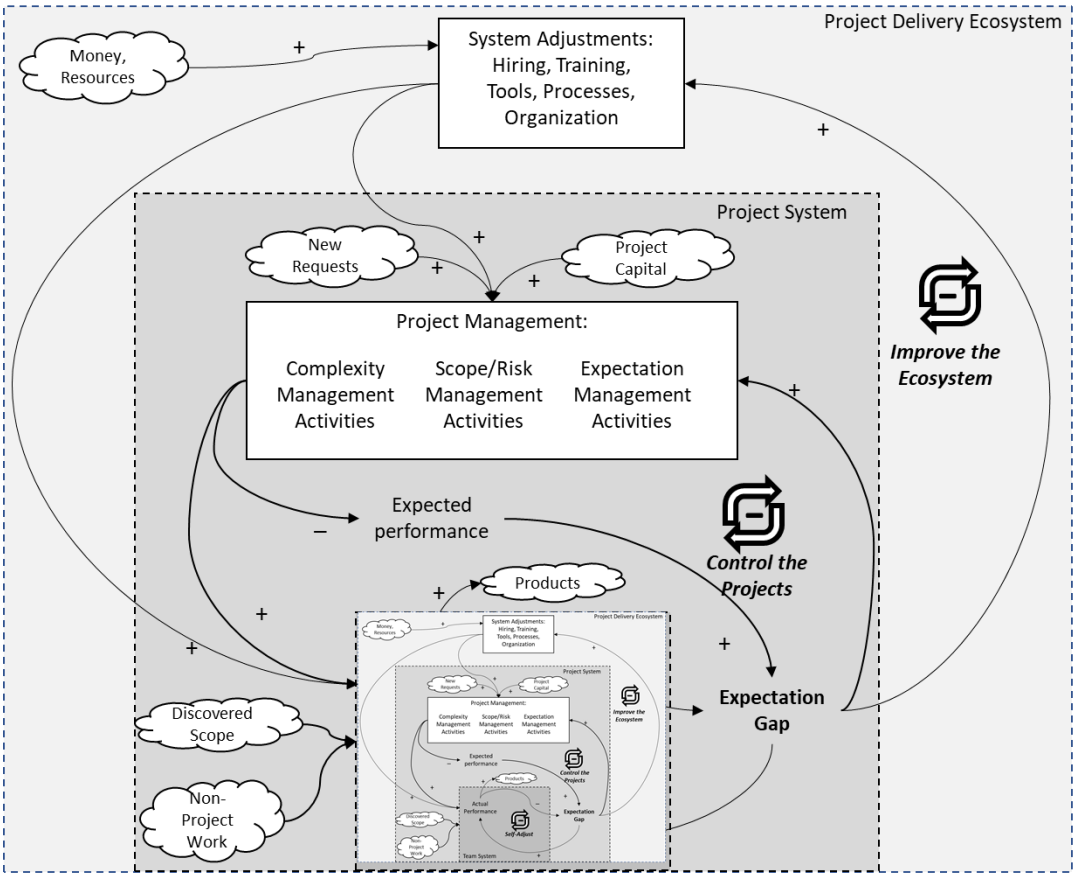


Figure 27: Recursive "System in a System" Nature of the Project Delivery System Model

3.3 Uses of the Project Delivery System Model

The project delivery system model's primary uses are expected to be the analysis of existing project delivery systems or as a framework that assists with either a new design or the improvement of existing project delivery systems. Every white square represents activities, and each arrow represents information or commands between the nodes of activity. The project system developer exercises this model by asking the following questions at each activity node.

- What are the goals?
- What are the activities?
- Who are the agents that complete the activities?
- What information is required, and where does it originate?
- What information or commands are delivered, and to what activity?

- How often does the activity need to occur?
- What tools and processes are needed?

As these questions are answered, more fidelity can be added to the model as more specific nodes and callouts of what information circulates. From here, more analysis and modeling techniques from the disciplines such as systems engineering, architecture, control theory, sociology, and business disciplines can be applied. The use of the model may be valuable as a static framework. However, system dynamics methods can also be applied to the model to create simulations of the project delivery system's performance, which allows for experimentation of various system architectures.

As an example, the model is applied to the classic Agile and Stage-Gate frameworks to identify gaps in the system when combining the frameworks to manage leadership expectations. First, the model is reduced to the causal loop of interest, which is the loop that includes the setting and managing of the expected performance. See Figure 28

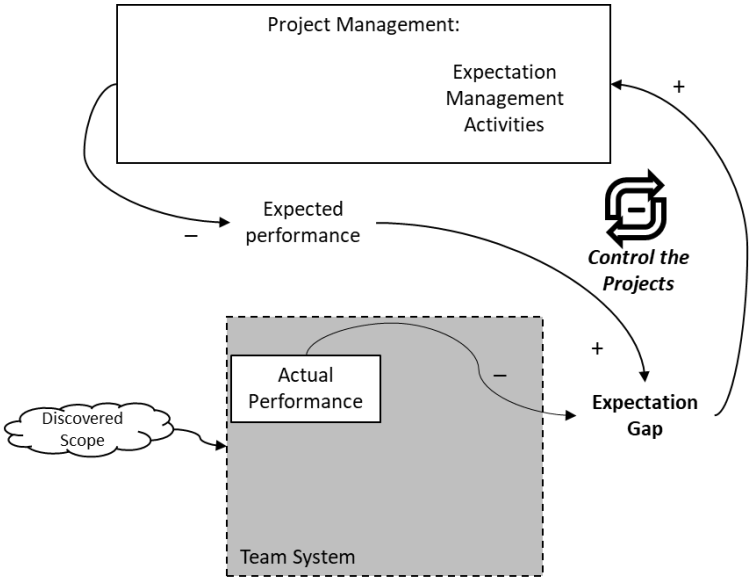


Figure 28: The Project Delivery System Model Focused Upon Expectations Management

Stage-Gate frameworks leverage a planning stage where the requirements are identified, and an execution plan is created with sufficient detail to create estimates of the amount of work and the time it will take to create it. This constitutes the creation of the initial value of the expected performance. Classic Agile frameworks do not provide long-term planning; thus, there is a gap in a hybrid system concerning defining a realistic expected performance that can be used to compare against actual performance.

Classic Agile frameworks expect that the team regularly and frequently delivers information on what scope has been completed. This data can be used by product management to create and refine the expected performance, but because such a concept is outside of the Team System, classic Agile

frameworks do not address this. Stage-Gate frameworks generally assume that the initial planning captures most of the total scope, along with some extra capacity, or slack, to account for small amounts of undiscovered scope. The framework does not account for the continuous flow of newly discovered scope.

Based on the gaps identified in the analysis of classic Agile and Stage-Gate frameworks, the following hypotheses are offered as potential remedies in the studied hybrid Agile Stage-Gate project delivery system.

- Agile teams engage in more up-front planning.
- More slack is added to agile team planning.
- Increased effort applied to change the Expected Performance.
- Ongoing scope and priorities management activities are added to the Stage-Gate part of the system.
- Delivery goals become more incremental, and therefore more fitting for Agile methods.

4 Research Method

This study will investigate the structures, methods, and behaviors that have evolved within several different product delivery systems to learn more about what emerges from within an Agile Stage-Gate project delivery system to manage and align leadership expectations to newly found scope. The investigation is expected to expose several different solutions for managing expectations, and the overall effectiveness, of each.

Focusing the project delivery system model upon the experimental questions yields the diagram found in Figure 28. The research seeks to identify what has evolved within the studied organizations' systems to manage the program leadership expectations when the teams discover previously unknown scope. The survey questions are designed to expose the processes, agents, and the data exchanged between the system nodes.

The chapter starts with some background regarding the subject groups for this study. It then discusses the methods used to collect the data.

4.1 Background Information on the Subject Organizations

The subject groups are project delivery systems responsible for delivering large complex mechatronic products requiring hundreds of engineers to develop. The primary project management method for these mechatronic systems leverage Stage-Gate frameworks, which was introduced decades ago when the products had no electronic or software content.

These mechatronic product development systems include a software development organization that produces solutions tailored to the specific mechatronic product. The software within these products exists to control and automate machine functions to improve the machines' performance and usability while they perform their duties. For each of these products, the introduction of electronics began slowly, requiring only small teams responsible for relatively simple control systems. Over the last two to three decades, the size and complexity of these products' software content have been growing exponentially, requiring proportional growth in the number of engineers dedicated to the product's software development needs.

Agile frameworks were first introduced into the software development organizations between 2003 and 2007. Once established, no further large-scale outside consulting or training activities continued after this initial introduction.

The Agile frameworks have not expanded beyond the software development efforts, including coding, testing, and calibration. For the subjects within this study, the software needs are determined mainly by the mechatronic machine's needs; thus, the customer to the software groups is the parent project developing the mechatronic product. The mechanical portions of the mechatronic product are still developed and managed using the established Stage-Gate methods.

4.2 Data Collection

Several project delivery systems were screened using informal screening interviews of the Stage-Gate project managers to identify the relative success the software development teams achieve with delivering to the parent mechatronic project's expectations. From this screening, three subject organizations with different organizational structures were selected for a more in-depth study.

Next, people who could provide the desired information about the project development system were identified and invited to participate. The participants include the leadership roles of the parent mechatronic project system. These are the people who possess the performance expectations that need to be managed as the project progresses. The participants also include leadership within the Agile project management organization that delivers software to the parent mechatronic project system. In all, the study included fourteen interviewees spread across the three project delivery systems.

The study pursued input from surveys of the software developers on the Agile software development teams. Candidates were identified by reviewing the organizational charts for the software delivery teams. Approximately 10-15 candidates were identified for each project delivery system.

The study participants are divided into three categories: the mechatronic project leaders, Agile leadership, and the software developers. Interviews were conducted with the project leaders and the Agile leadership. Software developers were questioned using a survey. The same questions were used for all the interviews, and similar questions formatted into the survey for the software developers. At least two responses were targeted for each category of respondents and each project system evaluated to provide corroborating information and supports the anonymity of the respondents. Appendix 1 includes the interview guide and the survey questions utilized for the data collection. As stated above, the questions intend to expose the activities, agents, and information used to manage project leadership expectations.

The interviews were conducted using video conferences and were about an hour in length. The interviews were recorded to augment and confirm the accuracy of the notes taken during the interview. The surveys were delivered digitally using an existing survey delivery software package available within each organization. This survey software is known to assure the anonymity of the survey participants.

5 Results

This chapter will illustrate the similarities and differences between project delivery systems from the structural, procedural, and behavioral viewpoints. The chapter will conclude by reporting the common themes and fundamental differences that the interviewees shared while explaining how each project delivery system operated and its strengths and weaknesses.

5.1 Survey Results

The data collection using surveys experienced several challenges that resulted in this portion of the data collection to be abandoned. The first issue was a general reluctance by the development teams' managers to allow the survey to be distributed to their teams. A second issue was a low response rate by the survey candidates. The first group of development teams for one of the project delivery systems yielded only three responses; thus, no conclusions could be drawn. The data is not presented here but included in Appendix 2 for reference.

5.2 Organizational Similarities

Figure 29 illustrates the general structure found in all three project delivery systems with highlighted areas identifying where the structures are notably different.

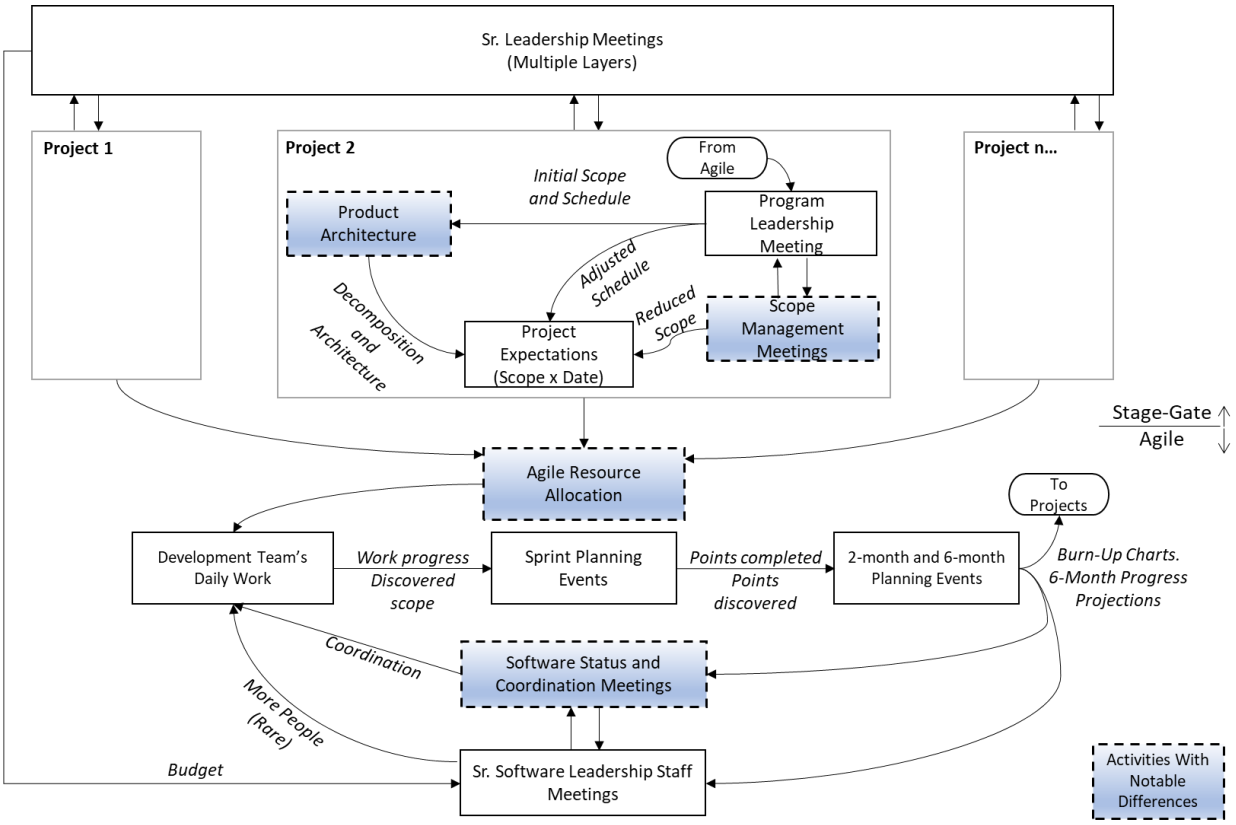


Figure 29: Structural Design of the Project Delivery Systems

Each project delivery system has planning and governance structures with many similarities. They use Stage-Gate process centered upon the delivery of mechanical systems. In each case, very few items have been added to the process that speak to managing software delivery. These additions are helpful, but the overall Stage-Gate processes are considered insufficient for the project delivery system's needs.

Each project delivery system utilizes hierarchical project governance, which is responsible for the scope and schedule decisions. This Stage-Gate structure is what the enterprise has used for decades. Each project delivery system has many projects in flight at the same time. Each project has a project-specific core team that provides governance and coordination between the main program functions, such as the different engineering disciplines, manufacturing, supply management, customer support, and marketing, to name a few.

Above this, a portfolio leadership team monitors and steers priorities across the many programs that are in flight. The highest level is the executive level, which includes the vice-president level of the enterprise. Minor adjustments to scope and schedule are handled at the lower layers. At the same time, changes that alter testing or product release plans are elevated to the executive level for approval.

Likewise, a hierarchical leadership organization exists for the Agile development teams within all three project delivery systems. This organization is parallel to the program leadership organization. Its role is to support and develop the Agile development teams instead of direct management of scope, schedule, or resources. This organization creates and maintains the Agile methods and ceremonies that are used

by the software development organization. It also monitors quality and efficiency and looks for adjustments and improvements to the tools, methods, and organizational structure to improve quality and efficiency.

For each project delivery system, the size of the software delivery organization is governed by mechanisms independent of any given project. Therefore, staffing does not scale according to the project's overall scope. This is a strategic decision that keeps the overall spend on research and development aligned to each company's overall performance. This causes staffing levels to be a slow adjustment that lags the need for any given project to a level that prevents it from being a useful lever for responding to scope discovery. The only fast-acting staffing lever available is to divert people from one project to another project.

The software development teams of the project delivery systems organize around a given technology domain of the mechatronic system. Thus, the backlog of any given software development team will have work coming from multiple projects. The software development teams report to managers and senior software leadership separate from the project leadership structure described above. The management level of the software delivery teams assists with coordination between the software development teams. They also have responsibility for providing the tools, processes, and training for the software development teams. The software leadership has brought Agile into each of these product development systems, and they have done it by "pushing" it upon the Stage-Gate program management structure.

All three project development systems utilize the role of a delivery lead. However, this role is a more recent addition to Project Delivery System C. Delivery leads are responsible for the software delivery to a specific mechatronic project. At the same time, product owners are assigned to a specific Agile software delivery team. The delivery lead works as the liaison between the Agile software development teams and the Stage-Gate program management teams, as illustrated in Figure 30. The delivery lead participates in the Stage-Gate leadership meetings and reports on the software delivery teams' progress and issues. They participate in all the Agile planning and coordination meetings to gather information and distribute guidance from the Stage-Gate program leadership. When needed, they also shoulder the responsibility of initiating the mitigation plans with the rest of the program stakeholders

Before delivery leads, these activities were shouldered by the product owners, but the product owners have the identification of requirements and team governance as their first responsibility. This caused the project management duties above to be left undone because the product owners did not have enough time to do both.

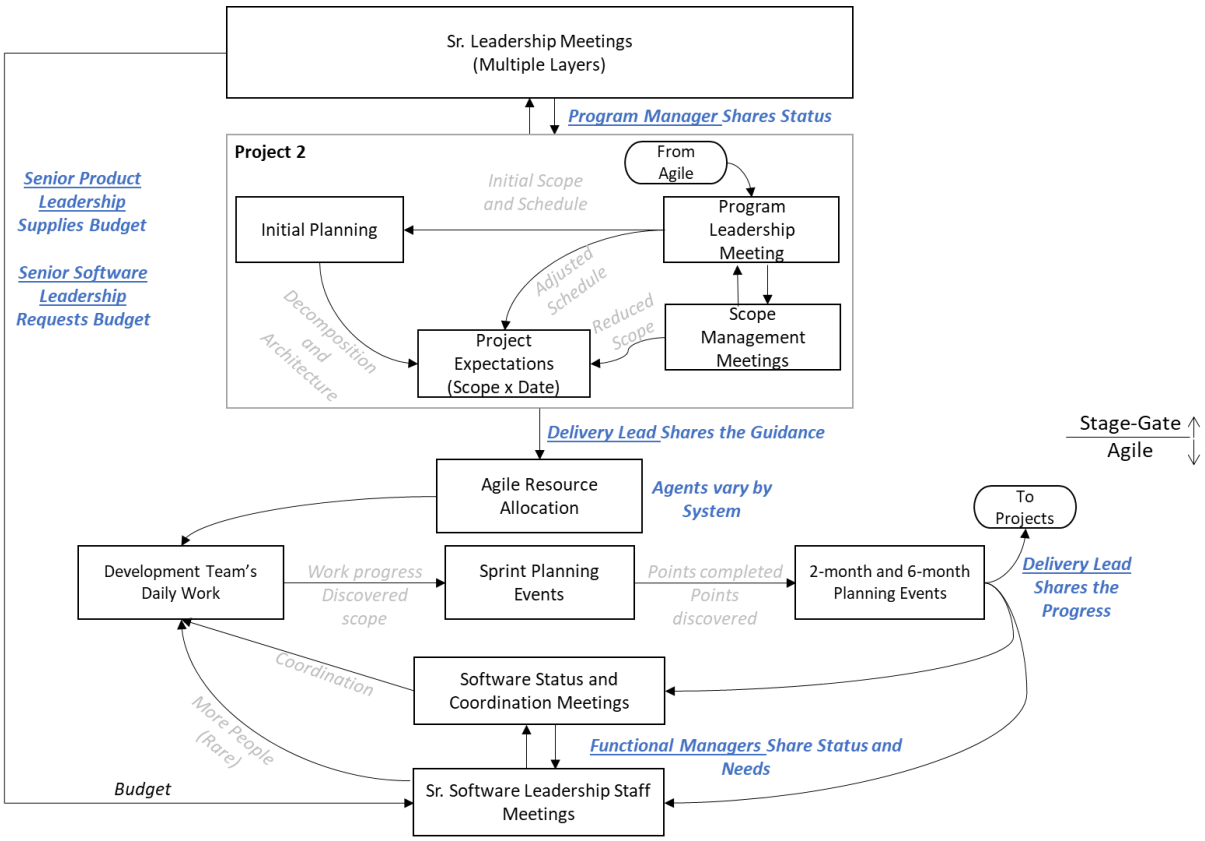


Figure 30: Key Roles within the Project Delivery System

The delivery lead facilitates communication from the Stage-Gate program leadership back to the software delivery teams. The venues used for this communication include informal communication as part of existing status and planning meetings with software delivery teams and team product owners and by the Agile software organization's software status and coordination meetings. Some of the groups also expect the delivery lead to share information regarding project leadership decisions at the delivery increment planning sessions.

5.3 Planning Stage Similarities

There are several similarities concerning the initial project planning. All three project delivery systems start with similar processes for digesting the marketing and product planning activities into a list of features to include in the project. The products delivered by these systems are new generations of an existing product, such as when an automotive company redesigns an existing automobile. The general architecture is an evolution of the previous architecture with new technologies and customer features added. Thus, the project scope is discussed as the list of changes from the previous product even though its magnitude can be significant.

The products produced by each of these project delivery systems first started as entirely mechanical devices, with projects introducing increasing electronic content in the last two to three decades. Therefore, the Stage-Gate project management processes focus on the delivery of mechanical

componentry. Classically, initial project milestones are defined by marketing and by the needs of the testing organization. Prototype builds are the dominant project milestones. They are set based upon hardware development processes. The physical system builds are completed to demonstrate progress and allow for testing and feedback.

The products adopted electronic content slowly, with modest functionality in the beginning. Formalized processes for managing software delivery were not required. Project management could plan the software scope with simple line-item callouts of functionality that was very often related to translating an operator command into a given actuator's motion to achieve the desired function and performance. Management did not have to concern themselves with the software because the mechanical systems harbored the risk.

The planning processes provided by the Stage-Gate methods have not evolved to provide for software scope planning. All the project delivery systems report that the initial project planning processes are currently insufficient for the complex electronic and software content that is now a substantial portion of the project's overall engineering effort. This results in unmeetable expectations concerning the delivery of software scope at cost and schedule. The recent efforts of Project Delivery System A related to improving their project planning and management system have begun to demonstrate improvement in setting realistic expectations. This will be discussed further in a later section.

Once a program is planned and underway, all three project delivery systems utilize Agile sprint and delivery increment planning and reporting. This is high fidelity planning and is expressed in the units of stories and story points. It takes the completion of many stories to add up to items of interest to the Stage-Gate leadership. The delivery leads process the high fidelity, story-based progress information from the planning events, the 6-month projections, and the burn-up charts into a message that is in the form that the project leadership can understand.

All project delivery systems report that the software development teams identify much of the scope discovery incrementally as small units of effort. As discussed in the paragraph above, scope discovery is at too high of fidelity for project leadership to process. One interviewee used the phrase "death by 1000 cuts" to explain the phenomenon. All the project delivery systems leverage the burn-up charts and six-month projections to identify and quantify the small units of discovered scope. The delivery lead is monitoring sprint and delivery increment results and the burn-up and six-month projections to form the communication of delivery risk and the need for mitigation that is at an aggregate level.

Virtually everyone interviewed within three product delivery systems reported dissatisfaction with software delivery because the planning phase of the Stage-Gate Process they use does not provide adequate and effective software planning. Several of the program managers interviewed stated that there is almost no mention of software deliverables at any program milestone reviews within the Stage-Gate program management process documentation. The Stage-Gate process utilized by these organizations do not seem to account for the fundamental need to reconcile the "Iron Triangle" of scope, schedule, and cost/resources for the project's software portion.

In the planning stage, the Stage-Gate processes used quantifies the scope of the project's design effort by quantifying the number of physical part designs that need to be released. This assessment of "part count" assumes a standard part design and release process and is known well enough to make reasonable effort estimates. This method does not readily apply to the software. The entire

mechatronic system's software only has a few actual "part numbers," and this in no way corresponds to the effort to design it. With no prescribed method for identifying software scope, the project plan that this process delivers regarding software scope is in the form of a bulleted list expressing the general desired functionality.

The interviewees in all the project development systems report that this is insufficient to make any reasonable estimation. When combined with the rigid staffing constraints discussed above, this leads to unachievable schedules and frustration with the software teams' overall inability to keep the schedule. Some also reported that the lack of explicitness in the software expectations upfront leaves the actual expectations open to change as the project progresses. One interviewee used the term "gold plating" to explain the phenomenon where the technical leads and developers add more capabilities and higher expectations to the deliverable than what was originally called for. Another interviewee cited the example where the initial expectations of the user interface have no definition to them at all when the project starts, leaving it entirely to iterative exploration as the work progresses.

5.4 Design Stage Similarities

Progress forecasting is provided by six-month projections of the anticipated software delivery using sprint and delivery increment planning events and the accompanying preparation meetings. These processes are based upon the Scaled Agile Framework discussed in the literature review. Delivery Increments for each project delivery system are eight weeks long and subdivided into four two-week sprints. Story point systems quantify the amount of work the backlog items will take. Each scrum team will make detailed plans for the next delivery increment, and less detailed plans for the next two delivery increments, resulting in a six-month forecast.

In addition to the six-month forecasts, all three project development systems utilized burn-up charts as a secondary progress reporting method. The milestones are usually set using the build dates of the mechatronic system. Agile leadership uses the burn-up charts to signal the need to research and mitigate the causes of falling behind with delivery. This reporting tool was not always utilized as a reporting format when presenting to the Stage-Gate program leadership because of insufficient understanding of what the document reveals. Some interview respondents identified that it was uncomfortable to see the "target" line moving. Presumably, because in their mind, the target, in terms of the delivered capability, is unchanging. Burn-up charts are in units of story points, which represents effort, not the delivered capability. Therefore, they are from a perspective that does not align with the Stage-Gate leadership's perspective.

The flow of software scope through the project delivery system during the design phase can be visualized with a modified version of the stocks and flows diagram proposed by Januszek. Figure 31 illustrates the flow of scope through the system.

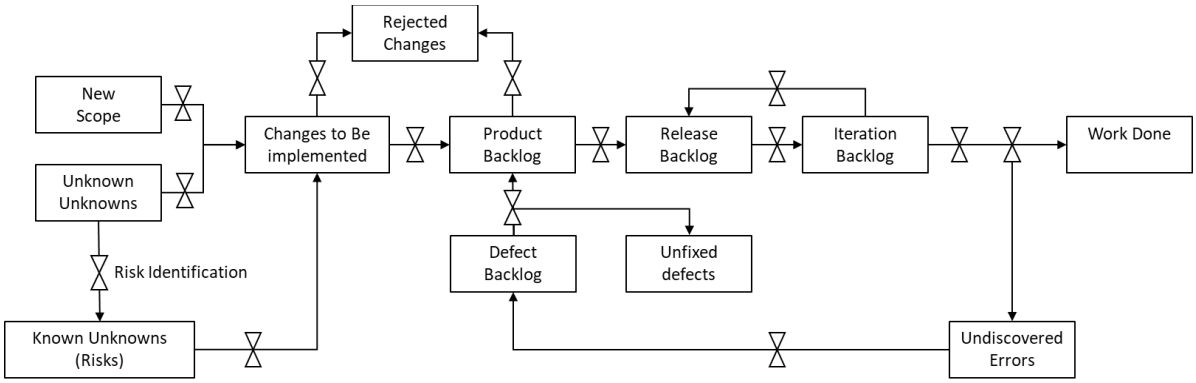


Figure 31: Stocks and Flows Model for the Studied Project Delivery Systems

The key differences from Januszek's model include acknowledging that bugs are not necessarily repaired. All three systems utilize a decision activity that evaluates the bug and determines the value of repairing it and the risks of leaving it. Sometimes low risk, low reward bugs are left in the product. A second difference is an explicit acknowledgment that new scope can enter the system as a request, which is evaluated as part of a scope management activity. Finally, like the discussion on bugs, it is possible that risks, once understood, may be left unmitigated if the risk is small and the effort to mitigate it is large.

5.5 Planning Stage Differences

While Project Delivery Systems B and C are still in a situation where their planning stage aligns with the discussion above, Project Delivery System A has taken on a significant process adjustment to resolve the issue of insufficient scope definition in their most recent projects. This organization has spent several years building and capturing the architecture and system decomposition of the electrical, electronic, and software systems included within the mechatronic products they produce. This architecture has high fidelity, down to software components that a team can generally complete in one or two delivery increments.

The most recent projects launched by Project Delivery System A are not allowed to exit the planning stage without explicit estimates of the software content and the effort required to deliver it. They are using this well-documented architecture to allow the planners to identify what software components will have to be changed and the magnitude of the change. The software plans are coordinated with the mechatronic hardware deliveries and testing schedule. The detail available within the software plans allows the delivery lead to engage with the rest of the project management team with data available to justify the need to negotiate scope and schedule.

The addition of an organization that creates and maintains the architecture for the embedded software was substantial and done primarily for managing the rapidly increasing complexity of the software contained within the products. The utilization of this work for project planning purposes is an additional benefit that satisfied the Stage-Gate and Agile interviewees.

5.6 Design Stage Management Differences

All three of these project delivery systems have recognized the need for more resources and events dedicated to the ongoing planning and coordination effort beyond product owners and the functional area managers that have existed since the first adoption of Agile. As discussed above, each has embraced the need for delivery leads. These delivery leads have different organizational structures that they work in and experience differing levels of authority bestowed upon them.

Product Delivery System A has bestowed substantial authority to the delivery leads regarding the ability to renegotiate initial scope and the scope content of delivery milestones. In the planning stages of the project, the delivery lead is responsible for building the program backlog using estimates provided by software development teams and software architects. As discussed above, the electronic architecture and decomposition are leveraged heavily for this. The backlog is considered 'sized' once the initial estimates are complete. The delivery lead then collaborates with the other subsystem leads for the mechanical systems to create an aligned plan.

The delivery leads in Product Delivery System A are supported by the rest of the organization structurally and culturally. Once development is underway, the delivery lead is monitoring progress and reporting risks. When progress falls behind the original plan, the delivery lead brings this to a scope management meeting. This is a working-level meeting that meets weekly. All development areas participate so that comprehensive delivery adjustments which account for the reality facing the software development team are made by all groups. Before this meeting, delivery leads are trained to proactively reach out to the affected groups and begin the mitigation planning so that the scope management meetings are more effective.

Product Delivery System A has engrained the need for reconciling scope to schedule into its culture. Stage-Gate program leadership has clear expectations that all development teams will adjust their plans collaboratively to maintain the delivery milestone dates. This includes reducing the capabilities of included features and removing less essential features from the mechatronic program. By far, this project delivery system is the most developed in these activities, with clear expectations, culture, processes, and events developed to fulfill the need to account for discovered scope and managing expectations.

The processes of Product Delivery Systems B and C do not support the need to reconcile scope to schedule as well as the most recent projects within system A. These groups have a culture that does not provide for the acceptance of the need to adjust the expected scope. When delivery leads in these systems need to make changes to the original scope or dates, there is less willingness by the rest of the development areas to adjust their expectations from the software group. This makes it difficult for the delivery leads to negotiate the necessary adjustments without active senior leadership support.

Project Delivery System B has made efforts to improve the project execution by adding explicit line-items related to software delivery to the Stage-Gate exit reviews. One example of this is an explicit call out of a content "Freeze Date" several months before the milestone. This has the effect of forcing the prioritization and management discussions to occur earlier. It also introduces some slack into the system

so that software delivery teams can address late-discovered bugs before the milestone. These line-items were recently brought back into their general process documentation.

5.7 Organizational Differences

While Project Delivery System A maintains a single set of project management meetings for all functional areas, Product Delivery System B has built a parallel multi-tiered, project leadership structure devoted to the software effort for all projects within the system. Most of the same leadership that participates in the Stage-Gate leadership meeting also participate in these meetings, but this meeting discusses all projects that are in-flight at the same time. The meetings exist to allow for the negotiation of mitigation actions across all in-flight projects to account for software falling behind the original plan. In this system, it appears that it takes higher levels of organizational leadership to enable the ability to adjust the original plan.

Project Delivery System C has introduced delivery leads only in the last year and only on one of the in-flight projects. Before introducing delivery leads, the product owners held the responsibility to try and negotiate mitigation plans. This proved challenging from a workload standpoint because the product owners were tied to the teams and not the projects. The software functional area manager was responsible for participating in the Stage-Gate project leadership meetings. However, these groups met separately from each other. There was no ability and little incentive for the different Stage-Gate leadership teams to support mitigation activities that may reduce their requested scope.

Before introducing the delivery lead, Project Delivery System C added a dedicated program manager to the Agile software development area. This person was of the same seniority as the Stage-Gate program managers, and this person has a high level of trust and rapport with those within the broader project delivery system. The software delivery teams have been planning at the delivery increment level for years, but no data was being processed or conveyed formally from these activities. This program manager was also able to justify the addition of a person to work with the scrum masters and product owners to collect and process the projections that came from the planning. This allows the Agile program manager to use it as leverage for negotiating changes to scope or schedule with program leadership.

The software program manager of Project Delivery System C leads a twice-weekly coordination meeting that the product owners, the delivery lead, and the software managers attend. This meeting is used for tactical coordination and news distribution regarding scope, schedule, or priority changes that the Stage-Gate leadership teams have made. Each team's progress is monitored against the delivery increment planning and the next milestone. This system also has a separate software calibration organization that is highly dependent upon software deliveries. This group meets weekly to coordinate the availability of the calibration staff and equipment with the software deliveries. This group is also useful for creating mitigation plans for software deliveries that are not holding to the original schedule, since this group is usually the first downstream users of the new software.

5.8 Differences with Managing Capacity Allocation

Another area where the project delivery systems differ is the method used to allocate resources to the various projects. All the project delivery systems have multiple projects being managed through Stage-Gate activities. Thus, the software delivery teams need to know how much of their capacity to dedicate to the various projects. Project Delivery System A has adopted a formal method for allocating capacity based upon the Scaled Agile framework. The delivery leads in this system all report to a single manager who manages the Agile processes and brings the Stage-Gate project managers together regularly to manage the distribution of the collective capacity of the Software development teams to the different projects. This is done in a lump-sum allocation, meaning that the project leaders agree to what percent of the total capacity they will receive, but not the details of who does what. Those decisions are left to the delivery leads and Product Owners to determine during their regular planning events.

Neither Project Delivery Systems B nor C interviewees identified a formal system for allocating capacity. Software leadership meetings serve to decide upon the allocation using project value, the relative schedule state, and budget spend. The results of which are communicated via delivery leads and functional area managers. Project delivery systems use timecard entries to record the amount of time spent on each project. Delivery leads and the software developers' managers communicate and coordinate the effort applied to each project according to the guidance from the leadership meetings.

5.9 Chain of Command Analysis

All three project delivery systems have introduced staff into the Agile software development organization that is explicitly responsible for executing the project management activities. This includes the delivery leads discussed above, and some form of a leadership role focused on the building and execution of an Agile project management method. Figure 32 represents the reporting hierarchy with the project decision-making chain of command overlaid upon it for Project Delivery Systems A and B. Figure 33 represents the same structures for Project Delivery System C.

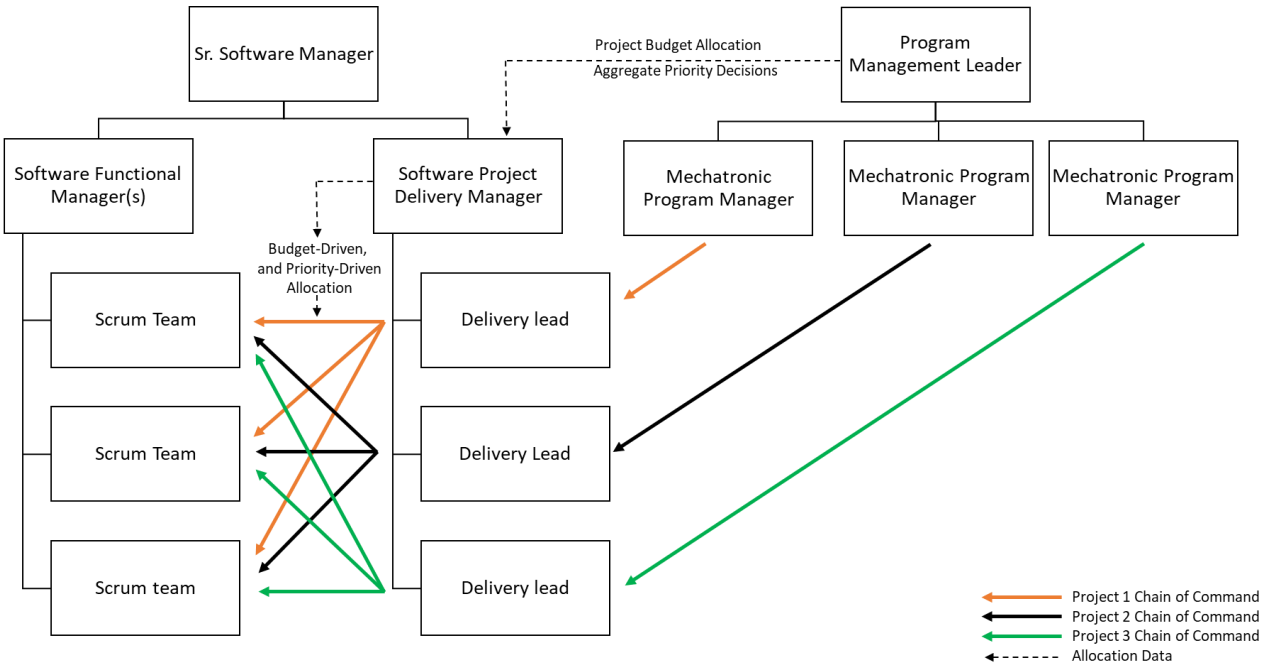


Figure 32: Organizational and Project Reporting Structures for Project Delivery Systems A and B

Project Delivery Systems A and B's reporting structure has the software functional managers and the delivery lead manager reporting to a senior software functional manager. This management chain is focused mostly upon maintaining and improving software development teams' capacity to deliver project scope and removing structural, organizational, and tool and process impediments that hinder this. The software functional managers have the software developers as direct reports, and the delivery lead managers have the delivery leads as direct reports. This structure allows each of these managers to focus on the needs of the roles that report to them, with the software functional manager focusing more on what software developers need for building software, and the delivery lead manager focusing more on what the Delivery Leads need. While not pursued as a direct question, many delivery leads and delivery lead managers provided comments that support this conclusion.

The direct reporting structure of delivery leads to a delivery lead manager reinforces the expectations of the delivery leads to focus on deliveries and the management of delivery expectations. The delivery lead manager is also the process developer for the Agile project management methods and leads the delivery increment planning events. Likewise, having software developers report to a person responsible for assuring that the software is built using best practices and is of high quality reinforces what the developer's primary focus is.

The projects' command hierarchy starts with a program management leader who has the program managers as direct reports. Program managers convene project management teams consisting of those responsible for managing the deliveries from their respective functional areas. The software delivery leads are the participants from the software functional area. The software delivery leads then interact

directly with the software delivery teams' product owners, sharing capacity guidance from the project leadership meetings led by the program manager.

When new scope is discovered, the primary path of action follows the project chain of command and not the reporting hierarchy. The product owners on the scrum teams report it to the delivery leads, who bring it to the program manager in a one to one conversation or during the project management team meeting. As discussed above, it is typical for the delivery lead to collaboratively formulate a mitigation plan with the other functional areas before the project management team meeting. If the issue requires trade-offs between the different programs, the item is brought to the senior project leadership meetings to formalize the negotiation between the programs.

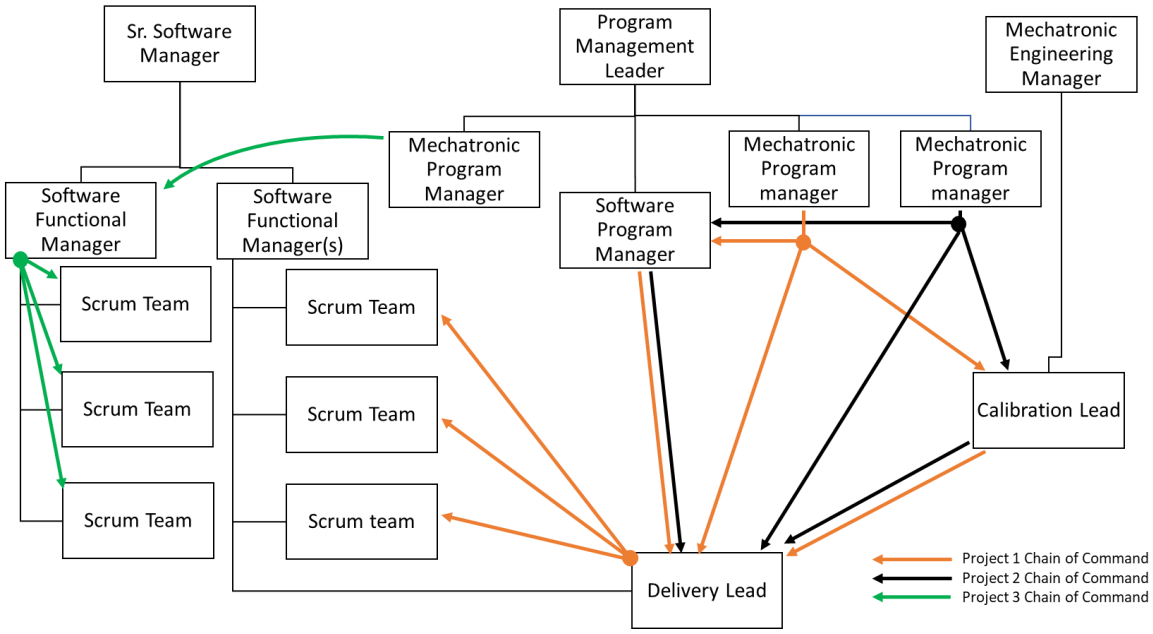


Figure 33: Organizational and Project Reporting Structures for Project Delivery System C

An analysis of the reporting hierarchy of Project Delivery System C reveals some additional complexities within the structure. One difference is that the delivery lead reports to the software functional manager. As discussed above, this organization has a software program manager instead of having a delivery lead manager. The software program manager serves a dual role in developing and managing the execution of Agile management practices and the project management activities that are classically associated with a program manager. The software program manager reports to the program management leader, along with the other program managers, and therefore benefits from the direct relationships and support of this functional area.

As discussed above, this project delivery system has a calibration team separate from the software development teams. This calibration community is very dependent upon the scope and timing of the software deliverables. This community is also a shared resource across all the mechatronic projects of this project delivery system. Because of these characteristics, a dedicated calibration lead provides coordination leadership between the calibrators, the software developers, and the mechatronic projects.

This organization's software delivery teams are split into two groups with separate software functional managers. The organizational split enables explicit project assignments for each organizational branch. There is also a technology boundary that aligns with this organizational split, where the scrum teams dedicated to the single project are using different electronics hardware for their project. The software functional manager completes the activities of both the software project manager and the delivery lead for the project assigned to those teams. This organizational split does make it harder to share work across the organizational divide, which can impede the ability to adjust to changing priorities.

5.10 Numeric Answer Results

Table 5 summarizes the interview participants' responses when asked about their perception of their project delivery system's overall effectiveness. The numeric range was defined as a one to ten scale, where one was equal to not effective at all, ten was perfectly effective, and five was functional, but with substantial issues. The average, range, and the number of responses are provided. The results are divided based on the project delivery system and which side of the Agile/Stage-Gate divide the interviewee worked in.

Table 5: Reported Overall Effectiveness of the Project Delivery System

Overall Software Delivery Effectiveness of the Project Delivery System							
	Stage Gate			**Agile**			Overall
	Average	Range	Responses	Average	Range	Responses	
System A	5.7	4 to 9	3	5.5	4 to 7	2	5.5
System B	4.75	3.5 to 6	2	5	5	1	4.8
System C	5	4 to 6	2	6.7	5 to 8	3	5.8
Overall	5.2	3.5 to 6	7	5.7	4 to 8	7	

Likewise, Table 6 provides the interview participants' responses when asked to provide their perception of how much the scope of the project grows during the time that the project is underway. It is important to note that most interviewees worked on different projects within the project delivery system. The results are categorized in the same manner as Table 5.

Table 6: Reported Scope Growth by Project Delivery System

Software Scope Growth by Project Delivery System							
	Stage Gate			Agile			Overall
	Average	Range	Responses	Average	Range	Responses	
System A	63%	25% to 100%	2	68%	35% to 100%	2	65%
System B	30%	10% to 50%	2	30%	30%	1	30%
System C	250%	250%	1	43%	30% to 50%	3	95%
Overall	114%	10% to 250%	7	47%	30% to 100%	7	

Table 7 summarizes the interview participants' responses when asked to rate the effectiveness of the methods that the project delivery system uses to keep the delivery expectations in line with the ongoing discovery of scope during the product design phase. Like above, the numeric range was defined at a one to ten scale, where one was equal to not effective at all, ten was perfectly effective, and five was functional, but with substantial issues. The average, range, and number of responses are provided, with the results divided based upon the project delivery system and which side of the Agile/Stage-Gate divide the interviewee worked. The results are categorized in the same manner as Table 5

Table 7: Reported Effectiveness of Scope Alignment Activities for the Project Delivery System

Effectiveness of Leadership Alignment Activities							
	Stage Gate			Agile			Overall
	Average	Range	Responses	Average	Range	Responses	
System A	7.7	7.5 to 8.5	3	7	6 to 8	2	7.2
System B	4.5	4.5 to 5	2	7	7	1	5.5
System C	5.5	5 to 6	2	7.7	7 to 8	3	6.8
Overall	5.9	4.5 to 7.5	7	7.2	6 to 8	7	

5.11 Interview Content Analysis

Each interviewee took the opportunity to explain their answers and discuss the general situation within their project delivery system. An analysis of the system attributes that the interviewees chose to discuss is summarized below. The information is split into the two categories of "Positive Topics" and "Negative Topics." Positive Topics were attributes that the interviewee presented as beneficial to the system. Negative Topics were attributes that were presented as detrimental to the system. Each category is presented from two perspectives. Figure 34 presents the most often identified positive topics and the frequency in which an Agile leader or a Stage-Gate leader chose to discuss it. Figure 35 presents the positive topics again, this time sorted by the project delivery system. Figure 36 and Figure 37 present the negative topics using the same pattern.

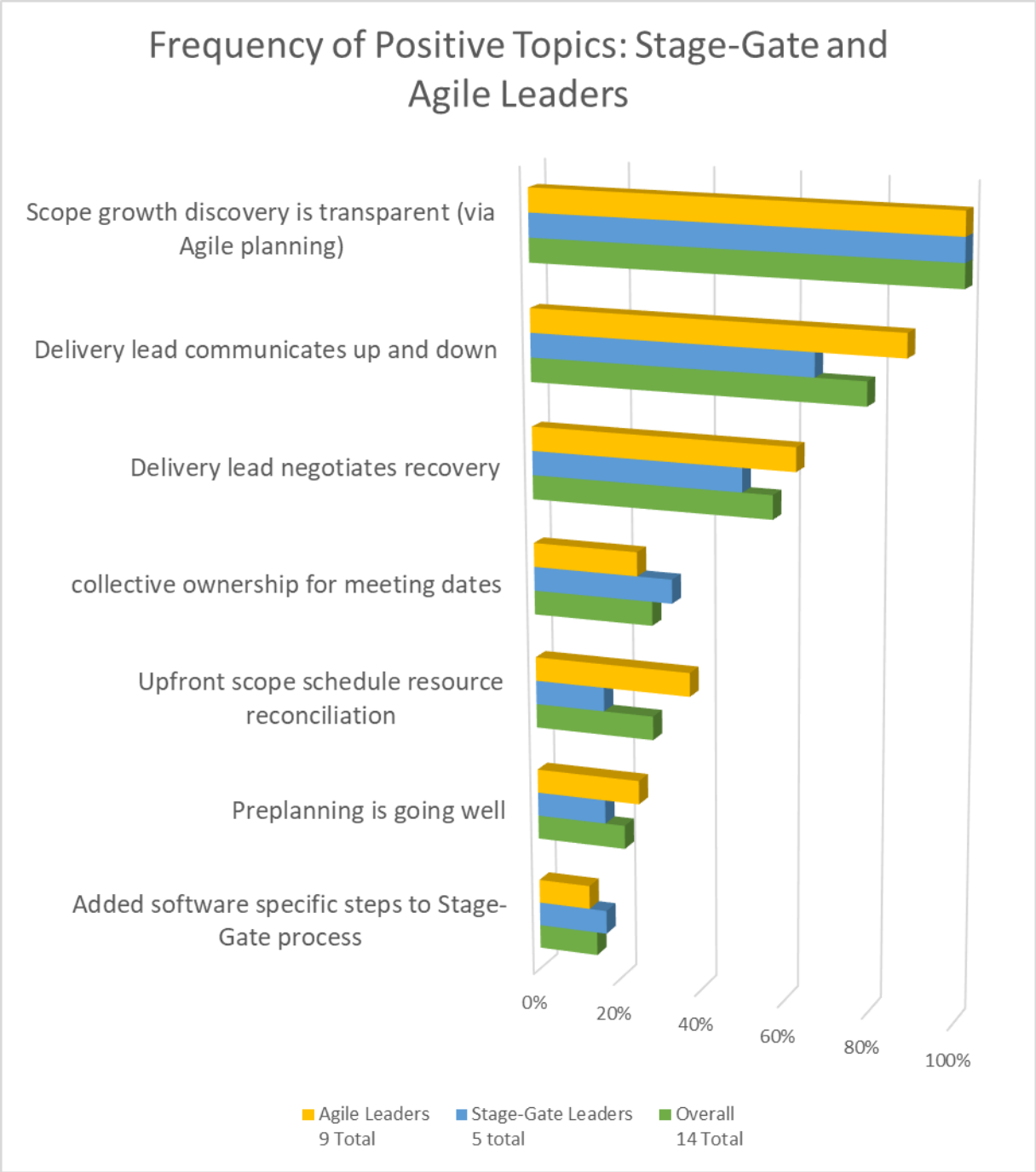


Figure 34: Frequency of Positive Topics: Stage-Gate and Agile Leaders

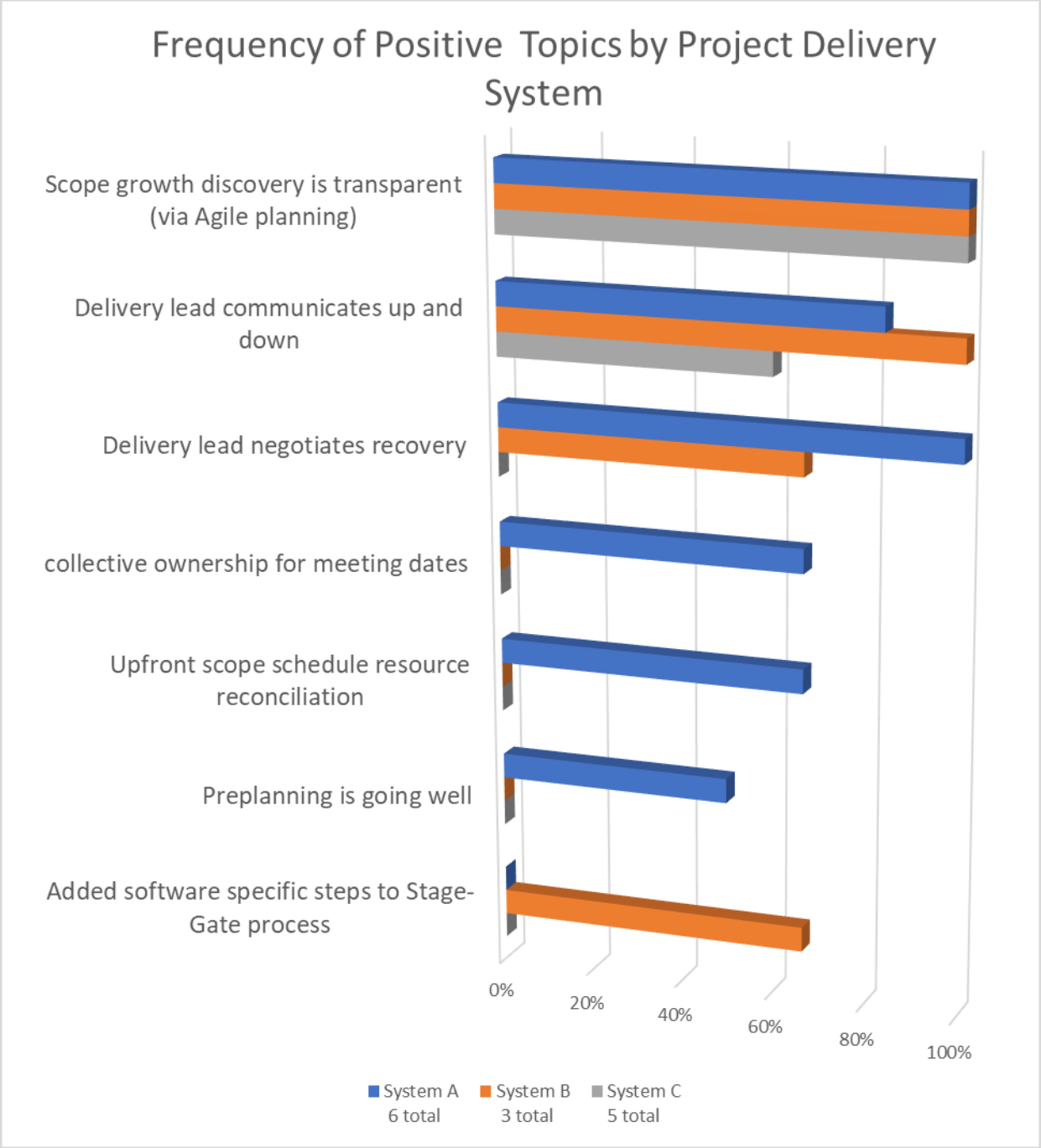


Figure 35: Frequency of Positive Topics by Project Delivery System

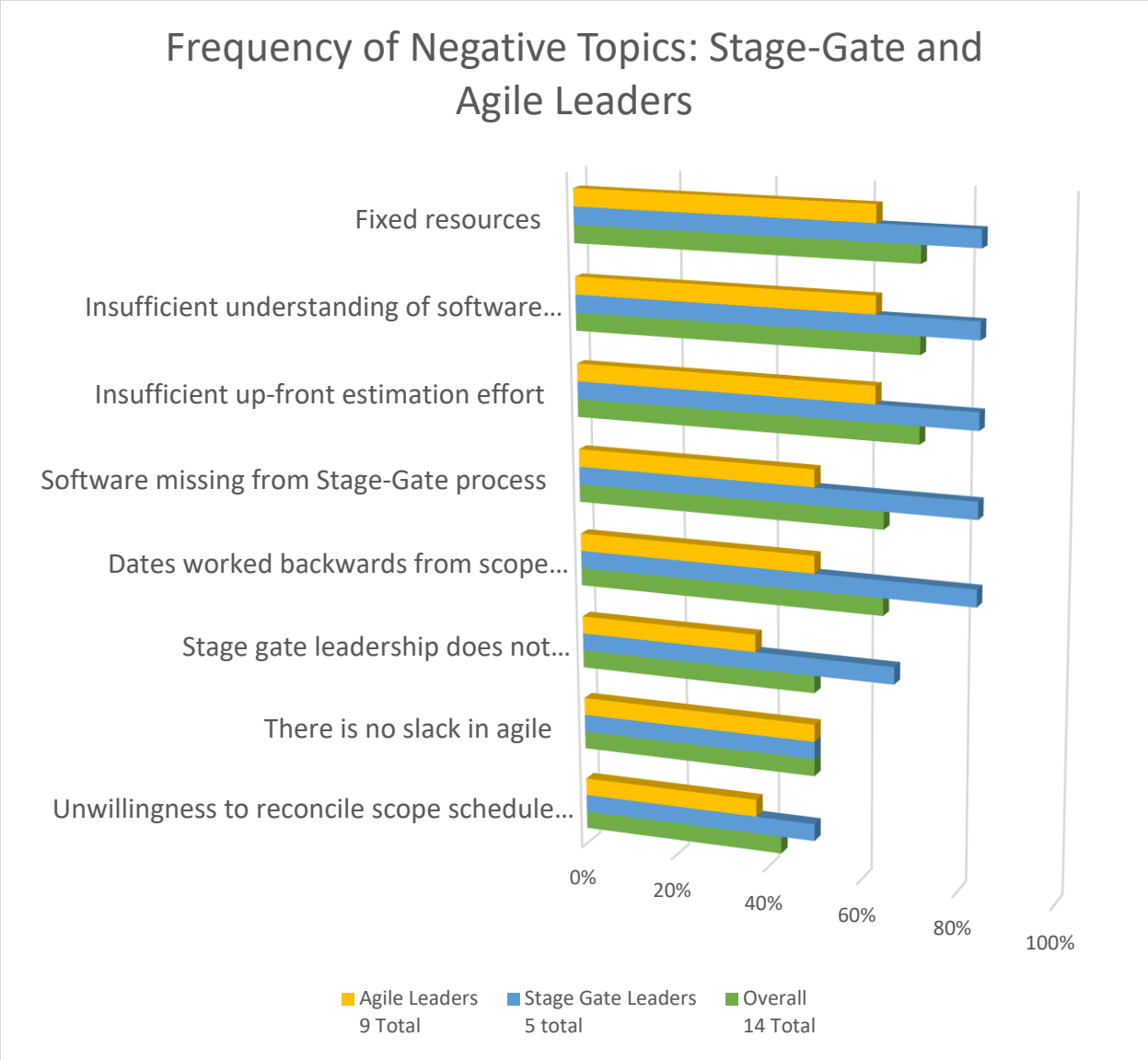


Figure 36: Frequency of Negative Topics: Stage-Gate and Agile Leaders

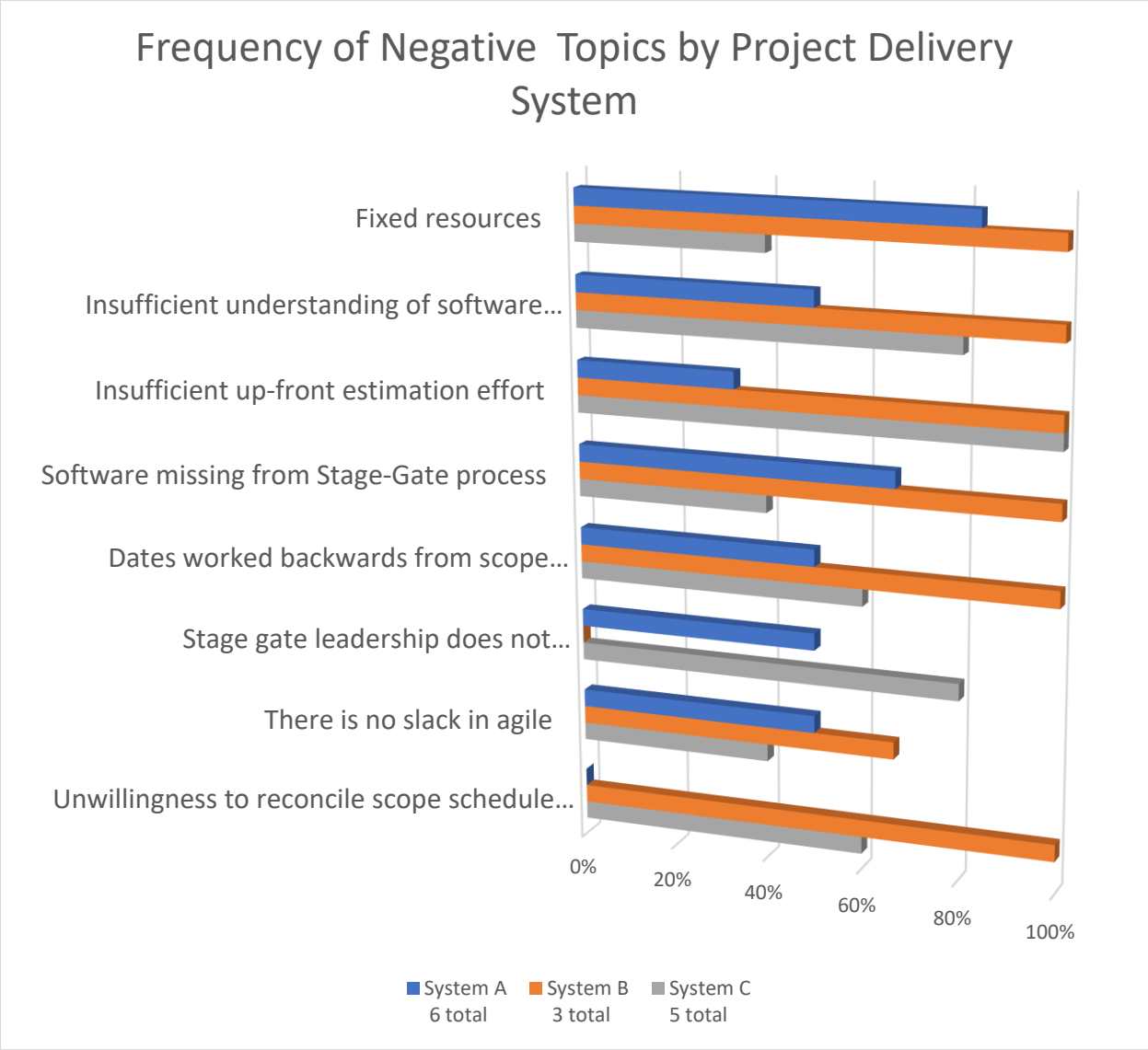


Figure 37:Frequency of Negative Topics by Project Delivery System

6 Discussion

The numeric data and the interview commentary indicate that Project Delivery System A seems to be on a path to the most successful integrating of Agile and Stage-Gate methods of the three systems studied. Systems B and C feel that they are improving but are still struggling relative to System A. System B expressed the most difficulty. The following discussion highlights what this research and analysis would suggest the key differences are.

6.1 Reconciliation to the "Iron Triangle"

Almost all interviewees expressed lower levels of satisfaction with the product delivery system's overall effectiveness than their satisfaction with the structures and processes that provide the ongoing alignment of the delivery expectations with newly discovered scope. When expanding upon the rationale for scoring the overall effectiveness so low, the interviewees cited the lack of effective project planning activities for the software scope as the root cause of the issue. This was consistent with both the Stage-Gate and the Agile leaders.

The interviewees' comments imply a fundamental gap within their processes for accounting for the concept of the Iron Triangle. Scope, schedule, and cost must be reconciled with each other to have a realistic project plan. This does not appear to be happening for most of the projects within these project delivery systems. All systems reported that staff levels are fixed using a process outside the project's control. Budget and staff allocation is based upon the revenue the mechatronic system generates. Scope and schedule seem to be set based upon the mechatronic system's non-electronic hardware content. Except for the most recent projects of Project Delivery System A, software scope is not reconciled to the schedule within the planning activities. This results in unattainable project expectations being set concerning the software's delivery and sets the project up to fail with meeting expectations right from the start.

Many interviewees cited insufficient effort to understand the scope of the software content that is part of the overall mechatronic project. They also call out that a formal estimation of the effort to deliver the content is also not done. Two reasons surfaced for this lack of planning: the lack of software planning activities within the Stage-Gate processes and insufficient resources applied to the planning effort.

The products delivered by these project delivery systems started as 100% mechanical products. Thus, the project delivery system and project leadership optimized to the needs of developing mechanical systems. The rapid rise of these systems' embedded content appears to have outpaced the ability of the project delivery system to adapt to the presence of such a large amount of software scope. Agile methods have been introduced by software delivery leadership in more of a grassroots approach. The mechatronic products' project leadership has not built an understanding of what needs to change to allow for more effective project management for these new software laden mechatronic products. This

has prevented the significant evolution of the project management processes and, therefore, the gap in software planning as part of the process's planning stage.

Why does there need to be a plan? Why can't the software simply be added when it is ready? Is the angst caused by having delivery milestones required for software only a remnant of a Stage-Gate legacy? Why not simply ship what is done? While Agile philosophies do make these arguments, the highly coupled relationships between the software and the hardware of these mechatronic systems force the need for planning. Hardware development has a different delivery cadence and different testing needs. The mechatronic systems these project delivery systems deliver cannot function without a high level of software scope present to operate all the hardware components present within the system. Thus, the software's delivery schedule must be planned out to align with the hardware deliveries so that the mechatronic system can be operated for testing and the sale to the customer.

As discussed above, Project Delivery System B has attempted to address some of the gaps with the project management processes by proposing additions to the overall Stage-Gate project management process. Although beneficial, interviewees within the Project Delivery System B acknowledged that they are insufficient to deal with all the issues.

Project Delivery System A has taken the most significant amount of action to fill the software planning gap with the project delivery system. This group has leveraged its architecture initiative to provide the knowledge necessary for improved planning. Before the launch of the most recent mechatronic project, this project delivery system established a documented functional decomposition and architecture for the software system. This is a graphical model using Model-Based Systems Engineering methods to identify the functions, the essential requirements for those functions, and the plan for the needed software components and subsystems.

This architecture was then used for an up-front estimating effort by the scrum teams. Before the estimation activity, the scrum teams completed training, which included reviewing the estimated versus actual effort during previous projects to deliver similar software solutions. In effect, this was an activity to calibrate the estimators' minds before delivering the new estimates. Data for the training activity came from the Agile sprint planning and delivery increment planning activities of the previous project.

The product decomposition and the estimating activity had allowed this project delivery system to build up the teams' backlogs with sufficient fidelity to project when elements of software scope were likely to be completed. The delivery lead could then use these estimates to negotiate milestone content with the rest of the non-software functional areas. This is being done in a culture that expects that the software plan will be explicitly delivered before the exit of the planning stage of the mechatronic project, even though it is not explicit within the process documentation. This level of planning is new to this organization, and the effectiveness of this planning effort is not available yet because the first project to use this is only now exiting the planning stage. However, both the Agile and Stage-Gate leadership interviewed involved with this project express satisfaction with the results to date.

Project Systems B and C have yet to improve their current process gaps with reconciling the requested scope to the schedule during the up-front planning. This is the most likely reason why these groups have the fewest positive comments and most negative comments about preplanning activities.

6.2 Stage-Gate Support of Agile

All three project delivery systems benefit from forecasting that comes from the regular planning events of Agile methods. The forecasting brings transparency to the project progress, and therefore the opportunity to act more quickly. As discussed in the introduction and literature review, the classic Agile methods do not provide guidance for how to act upon the data. It merely becomes the responsibility of "The Customer," but with these project delivery systems, the customer of the software is the mechatronic project. Therefore, the Stage-Gate project management methods must respond to the feedback that Agile systems provide.

Culture is the key difference between the project delivery systems regarding the management of discovered scope. All groups report that Stage-Gate leadership does not understand Agile well enough. However, Project Delivery System A reports that the non-software functional areas have been directed to support the need to adjust their requested software scope to maintain milestone dates. They have also been coached to accept the schedule forecasts coming from the Agile planning events and to work with the delivery leads to identify the mitigation plan.

This cultural shift has enabled Project Delivery System A to create an efficient system for adapting to discovered scope. Recall Figure 32 above. Culture allows for the delivery leads to be more effective with creating mitigation plans without as much need for support from higher-ranking leadership. A single weekly meeting with the project core team is all that is required to keep most sources of scope additions managed. The delivery lead has the authority and backing of leadership to make the necessary adjustments compared to the other project delivery systems.

Project Delivery System B and C have additional structure and roles within their management systems to compensate for the lack of a culture that accommodates the need to keep scope aligned to the schedule. Project Delivery System B has a duplicate set of project leadership meetings with all project leadership and functional area managers participating. These meetings provide the forums for negotiation of scope and schedule by the more senior levels of leadership. This method appears to require more collective effort by the organization than Project System A requires creating mitigation plans.

Project Delivery System C compensates for the lack of cultural support for Agile methods with the software program manager role. As discussed above, this role reports to the same leadership as the mechatronic program managers and possesses the same hierarchical rank in the organization. The delivery lead is responsible for identifying and recommending the mitigation plan that will compensate for the discovered scope but often must rely upon the software program manager for leverage with the other functional areas. Project Delivery System C also lacks any formalized methods for coordinating priorities across the multiple mechatronic projects in flight. It is the software program manager that has sufficient leverage to procure concessions from the different programs. Before having this role present, the product owners and the software functional area manager shouldered the competing requests of the different programs directly and did not have sufficient organizational leverage to allow them to procure concessions effectively.

6.3 Implications

This section explores the implications of the findings of the literature review and the interviews on what a project delivery system could do to continue to improve. This will be done by proposing improvements to the project delivery system and a rationale for these improvements.

Let us consider Project Delivery System A as a baseline starting point to build the improvements upon. Like all the project delivery systems studied, this system has embraced both sprint and delivery increment planning activities to provide delivery forecasts and visibility to added and discovered scope. Like the other systems, this system has added roles dedicated to managing these planning events and delivery leads who are responsible for formulating any needed adjustments. This system already has the vital cultural traits in place and supported by the top of the organization that drives the collaborative reconciliation of scope to schedule. Thus, the organization is already efficient with its project chain of command and the meetings that support rapid adjustments to the project plan.

The classic Agile frameworks require small units of work to be packaged ahead of time and fed into the software development teams' backlog. They do not account for who or how these backlog items are created. Project Delivery System A has provided for this need by leveraging the efforts to define and capture the product functional decomposition and physical architecture. The decomposition and architecture allow the product system to be subdivided into modules assigned to individual teams. The decomposed modules' interface plan allows the teams to coordinate their activities and their project plans more readily. The software product owners have been able to leverage this information to improve their backlog's fidelity and their estimate's accuracy because they have visibility to a complete dataset of the intended scope.

The nature of software introduces high levels of uncertainty with defining what is needed. This makes estimating very difficult. Another item that needs to be accounted for more completely is that there is no slack in Agile planning. The planning activities within all the Agile frameworks reviewed result in all the capacity being utilized. Combining that with the tendency to underestimate the effort required and the inherent uncertainty of the requirements, makes the need for mitigation activities a certainty. This is what upfront planning should account for to improve planning within these project delivery systems.

The first adjustment proposal is to expand the up-front planning activities to proactively account for the certainty of scope overruns with respect to schedule. Two methods should be considered. The first leverages the concept behind what the Scaled Agile Framework prescribes, which creates explicit prioritization of the features that are part of the project. Since the project will certainly be faced with reducing scope at some point, the planning stage should confront the issue up-front with explicit prioritization agreed upon by the software and non-software functional areas. This creates an understanding up-front of the reality that a feature may have to be dropped from the scope. Thus, the contingency plans can take advantage of the flexibility that comes with acknowledging this up-front. Upfront contingency planning also reduces the emotional reaction to enacting the plan because the path is already understood.

The second adjustment proposal is an extension of the first one but is likely more difficult and more nuanced. The first adjustment prioritized scope at the feature level and called for contingency planning that would remove features from the scope. This proposal is to create scaled versions of the desired

feature as contingency plans that proactively exchange effort for scope. In effect, this an "Intra-Feature Scaling and Contingency Plan." Consider the analogy to the "80/20 rule," which asserts that you can get 80% of the benefit from 20% of the work. This proposal calls for the software functionality requesters to determine several levels of deliverable capability for the feature, each with a different level of effort and a different level of expectations for what/how the feature will do/perform. This increases the options that can be pursued when faced with scope overruns while executing the project.

An additional benefit to this second proposal is that it allows the project to achieve a testable product more quickly. This will better support the testing cadence of the non-software parts of the product because the software has at least a baseline functionality that enables a sizeable amount of system testing. It also enables the opportunity to decide that the product has reached a level of value to the customers where it can go to production without all the planned scope present. This is a valuable opportunity due to the potential to reduce the development length. It also provides better visibility into what is being asked for and thus provides a management lever for managing scope creep or the "gold plating" of features by including things of little incremental value to the product overall.

Table 8: Stage-Gate Process Additions to Support Agile Project Management

Stage-Gate Process Additions to Support Agile Project Management	
Planning Phase (Exit Criteria)	Software Functional Decomposition
	Software Physical Architecture
	Prioritized Software Feature Backlog
	Feature Level Contingency Plans
	Intra-feature Scaling and Contingencies
	Feature-to-Team Assignments
	Team Level Backlogs with Intra-Feature Scaling Options
Design Phase	Agile Planning Event Support
	Regular Schedule Forcast Reviews
	Regular Scope Management Meetings
	Ongoing Architecture and Decomposition Manitenance
	Ongoing Capacity Allocation Management
	Ongoing Contingency Plan Maintenance Activities

Finally, all the established and proposed methods need to be incorporated into the Stage-Gate process documentation, and the rationale behind the added activities must be trained. Most interviewees identified insufficient coverage of software activities within the Stage-Gate project management processes as the primary contributor to their low assessment of the overall project delivery effectiveness. Many interviewees also reported insufficient knowledge of Agile and how it works within the project delivery system as a problem. The size, complexity, coordination, and codependence between the hardware and software functions of this organization's mechatronic products drive the need for Stage-Gate methods to provide the planning required for the use of Agile for software development. Stage-Gate leadership must be trained so that they can support those needs.

Table 8 summarizes the additions to the Stage-Gate methods identified and proposed within this body of work. The additions to the planning phase can be activities that should be completed as exit criteria of the stage. Additions to the design phase are different in that they are ongoing activities that repeat on a regular cadence within the stage. The activities allow the project delivery team to more respond to risks of new or discovered scope more readily.

7 Conclusions

7.1 Research Conclusions

Software development, by its nature, is full of uncertainties. It is difficult to understand and express what is expected for the solution. There are many technical unknowns about how to deliver the solution. Along the way, the consumer of the solution may discover new expectations that were not expressed initially. These events result in scope being added to the project that was unaccounted for in the initial project planning. This paper refers to this as discovered scope.

Stage-Gate project management systems fail to acknowledge the nature of software development because it assumes that the requirements can be wholly understood up-front and satisfies this with a thorough planning phase. Discovered scope is an exception to the normal process and is treated as an anomaly. There is also an assumption that slack exists in the estimates that account for small amounts of discovered scope.

In contrast, Agile methods assume that the requirements cannot completely be determined up-front and instead react to new learning in-flight. However, this creates issues when using Agile methods as part of the development of mechatronic products, which have heavy scheduling dependencies between the hardware content and the software content. The classic Agile frameworks of Scrum, Kanban, and XP do not provide methods to coordinate and manage the delivery schedule of hardware and software scope.

Furthermore, there is an implicit assumption that the software is mostly free from hardware dependencies within the classic Agile methods. The final product delivered to the customer is the software itself. Complex mechatronic products are products that utilize software as a component of the larger mechanical system. Consumers of mechatronic systems are fulfilling needs centered around the mechanical attributes and may not care or even realize that software plays a role in the system. Thus, the software is a layer removed from the actual customer, and a surrogate is required. The actual needs of the software are derived from what the mechanical systems need to execute their functions.

This study concludes that mechatronic system development utilizing Agile methods for software development stands to benefit from a superseding Stage-Gate method that provides the planning and coordination between the software and non-software deliverables of the project. Classic Agile methods provide tactical planning for the design stage of the typical Stage-Gate process. The efficient execution of these methods relies on a modified set of Stage-Gate deliveries and activities to manage priorities in the presence of discovered scope. The planning stage of Stage-Gate provides a method to process the software requests into Agile stories using architecture and functional decomposition of the project into the small autonomous units that the Agile methods require for efficient execution.

This study identified many effective process, organizational, and cultural adjustments that have been successful within the project delivery systems studied. This includes embracing a culture that proactively responds when faced with the reality that all the project scope cannot be known upfront.

This also includes up-front planning that leverages the explicit capture of a software functional decomposition and architecture and creating multiple layers of scope prioritization and mitigation plans.

Once the design stage is underway, the systems studied demonstrated several ongoing scope management activities and chain of command structures of varying effectiveness. This includes Agile's focus on efficient delivery, while Stage-Gate focuses on processing requests into an actionable set of scope to be delivered and providing governance all scope delivery efforts. It includes Stage-Gate activities that digest and respond to the valuable and rapid feedback that the regular software planning and forecasting activities deliver. It includes establishing scope management activities to vet the need to pursue new or newly discovered scope and facilitating the creation of mitigation plans when scope delivery is not meeting the schedule. It also includes capacity allocation activities that explicitly identify how much of the collective software development effort is to be used for the various projects in flight.

This study concludes that there is value to the addition of dedicated delivery lead roles that facilitate communication and coordinate response activities when hidden scope is discovered. The addition of a role responsible for developing and leading the execution of Agile management practices is also valuable.

This study concludes that there is a benefit to having the entire project delivery system embrace the reality that scope discovery and scope additions will occur and must be addressed promptly and collaboratively by all functional areas. This allows the project delivery system to evolve to a more efficient organizational structure and a suite of processes.

Because of the inherent lack of slack within Agile planning, the nature of humans to under-estimate the size of the project effort, and the reality of new scope being discovered, this study proposes that the planning phase should include proactive project-wide and intra-feature contingency planning. This expedites project recovery and reduces the angst that comes with short-notice mitigation activities.

This study also found that executing an explicit Agile Stage-Gate hybrid project management process requires training as a necessary activity to allow for the Stage-Gate and Agile practitioners to understand how to provide for the needs of these combined methods.

7.2 Limitations

There are several limitations to the extensibility of the conclusions of this study. While each of the project delivery systems studied operate independently of each other, they build from the Scaled Agile Framework concept. It is unclear if the results would be different if a different Agile method would have been used by some or all the subject systems.

The mechatronic products developed by these project delivery systems may or may not have unique relationships between hardware and software, which may hinder the application of these findings onto other systems. This study did not attempt to identify any such relationships or compare them to other mechatronic products.

The interpretation of the Interview results is subject to the author's biases, who is a systems engineer by training and has his own experiences with the use of Agile Stage-Gate hybrids.

All the information from the interviews are subjective, including the numeric responses and subject to the interviewees' biases. Whenever possible, interviews of different individuals in the same or similar roles were completed to compare the responses to help mitigate the chance for biases. In addition, the number of interviews captured within each project delivery system could allow for the biases of a single or small group of individuals to impact the results.

7.3 Future Work

As an expansion to this body of work, the following activities could be considered.

A follow-up study with one or all these project delivery systems could be pursued, should any of them choose to apply some or all this study's recommendations. There are opportunities to evaluate the repeatability of the success that one of the project delivery systems had when the treatments are applied to the other two systems.

The modified systems dynamics model of Januszek could be expanded upon to create a set of simulation results to identify the repercussions of the proposed adjustments.

A repeat of this study or a comparison of the results of this study to a similar study on additional organizations could bring additional support to the conclusions of this study. This study could also be repeated with a group explicitly using a different Agile framework as their starting point to see if this impacts the results.

The software development groups within these organizations pursued Agile methods by having the software organizations push it into the project management practices. This bottom-up approach could be part of why the non-software parts of the organization have been slow to adopt any changes. Comparing an organization that pursued the Adoption of Agile Stage-Gate hybrids from the top-down would be an interesting comparison.

Finally, this work, the proposed future work above, and the existing work of others may enable a future activity of creating a more explicit and specific framework for Agile Stage-Gate project management systems to be proposed.

References

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, M. Fowler, R. Martin, S. Mellor, D. Thomas, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, K. Schwaber and J. Sutherland, "Manifesto for Agile Software Development," 2001. [Online]. Available: https://www.agilealliance.org/agile101/the-agile-manifesto/. [Accessed 7 June 2020].

[2] Project Management Institute, A guide to Project Management Body of Knowledge (PMBOK Guide), Newtown Square: Project Management Institute, 2017.

[3] B. Moser, "Scope Patterns for Projects Modeled as Sociotechnical Systems," *The Journal of Modern Project Management,* Vols. January-April 2017, pp. 118-122, 2017.

[4] D. Leffingwell, Scaling Software Agility, Upper Saddle River, New Jersey: Addison-Wesley, 2007.

[5] F. Brooks, The Mythical Man-Month, Boston: Addison-Wesley, 1995.

[6] J. Lyneis and D. Ford, "System dynamics applied to project management: a survey, assessment, and directions for future research," *Systems Dynamics Review,* vol. 23, pp. 157-189, 2007.

[7] S. Januszek, "Master's Thesis: Analyzing the Impact of Agility Enabling Team Structures on the Performance of Product Development Projects," Massachusetts Institute of Technology, Boston, 2017.

[8] A. Thayer, A. Petruzzelli and C. McClurg, "Addressing the Paradox of the Team Innovation Process: A Review and Practical Considerations," *American Psychologist,* vol. 73, no. 4, pp. 363-375, 2018.

[9] J. Stjepandic and W. Verhagen, Concurrent Engineering in the 21st Century, Switzerland: Springer International Publishing, 2015.

[10] J. Galbraith, "Organization Design: An Information Processing View," *Interfaces,* vol. 4, no. 3, pp. 28-36, 1974.

[11] K. Forsberg, H. Mooz and H. Cotterman, Visualizing Project Management, New York: J, Wiley and Sons, 2005.

[12] B. Blanchard and W. Fabrycky, Systems Engineering and Analysis, Upper Saddle River: Prentice Hall, 1998.

[13] Federal Highway Administration, "Clarus Concept of Operations," October 2005. [Online]. Available: https://web.archive.org/web/20090705102900/http://www.itsdocs.fhwa.dot.gov/jpodocs/repts_te/14158.htm. [Accessed 8 June 2020].

[14] B. Boehm, "A spiral Model of Software Development and Enhancement," *Computer,* vol. 21, no. 5, pp. 61-72, 1988.

[15] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, M. Fowler, R. Martin, S. Mellor, D. Thomas, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, K. Schwaber and J. Sutherland, "12 Principles Behind the Agile Manifesto," Agile Alliance, 2020. [Online]. Available: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/. [Accessed 8 June 2020].

[16] K. Schwaber and J. Sutherland, "The Scrum Guide," 2020. [Online]. Available: https://www.scrumguides.org/scrum-guide.html. [Accessed 8 June 2020].

[17] Scrum.org, "What is Scrum?," 9 June 2020. [Online]. Available: https://www.scrum.org/resources/what-is-scrum. [Accessed 9 June 2020].

[18] D. Anderson, Kanban: Successful Evolutionary Change for Your Technology Business, Sequim Washington: Blue Hole Press, 2010.

[19] K. Beck, Extreme Programming Explained (Second Edition), Boston: Addison-Wesley, 2005.

[20] D. Wells, "Extreme Programming: A Gentle Introduction," 8 October 2013. [Online]. Available: http://www.extremeprogramming.org/. [Accessed 9 June 2020].

[21] D. Leffingwell, R. Knastner, I. Oren and D. Jemilo, SAFe Reference Guide 4.5, New York: Pearson Education, 2018.

[22] Scaled Agile, Inc., "SAFe," Scaled Agile, Inc, 2020. [Online]. Available: https://www.scaledagileframework.com/. [Accessed 9 June 2020].

[23] The LeSS Company B.V., "LeSS," The LeSS Company B.V., 2020. [Online]. Available: https://less.works. [Accessed 10 June 2020].

[24] The LeSS Company B.V., "LeSS Huge," 2020. [Online]. Available: https://less.works/less/less-huge/index.html. [Accessed 10 June 2020].

[25] R. Cooper, "Agile-Stage-Gate Hybrids," *Research-Technology Management,* vol. 59, no. 1, pp. 21-29, 2016.

[26] A. Hamilton and E. Bergmann, "Agile & Waterfall: living together in perfect harmony!," 1 August 2013. [Online]. Available: https://www.agilealliance.org/resources/sessions/agile-waterfall-living-together-in-perfect-harmony/. [Accessed 10 June 2020].

[27] S. Vergini, "Burn Down vs Burn Up Chart," Project Management.com, 2020. [Online]. Available: https://www.projectmanagement.com/blog-post/40731/Burndown-vs-Burnup-Chart. [Accessed 19 August 2020].

Appendix 1: Interview Guide and Survey Questions

Interview Guide

1. How effective are the project management methods of this organization with respect to managing software delivery?
2. How does project leadership assess if the software development work is meeting expectations?

   While the software development teams are doing their work, they can discover more things that can, should, or must be done in to deliver to the requirements of the system.

3. Over the length of the project, how much more work is discovered and added to the original estimates?
4. How much of this discovered scope is shared with project leadership?
5. What is the method for sharing?
6. When new scope is discovered by the team, what methods does the project leadership use to adjust the expected schedule, scope, and/or resources to allow for the completion of the additional work?
7. How are these changes expressed by the project management?
8. How often do these activities occur?
9. How effective are these mechanisms with keeping the scope, schedule and resource expectations aligned with the discovery of additional work?
10. What activities do you find particularly effective? What barriers exist that prevent this from occurring?

Survey

Each question will have a 7 level range from Bad/Ineffective to Good/Effective. A few questions will have an open section where participants can add more information.

1. How effective are the project management methods of this organization with respect to managing software delivery? While the software development teams are doing their work, they can discover more things that can, should, or must be done to deliver to the requirements of the system.

   While the software development teams are doing their work, they can discover more things that can, should, or must be done in to deliver to the requirements of the system.

2. On average, how often does this occur within your team?

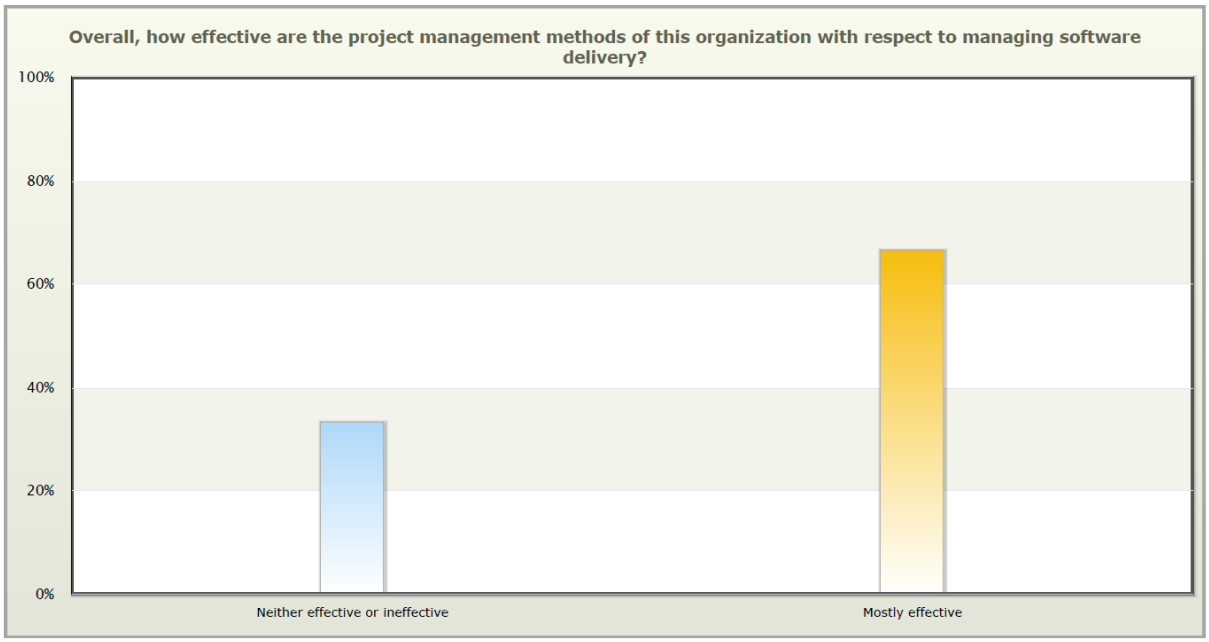
    a. Daily, Many times per week, Once per week, Many times per month, about once per month, about once per Quarter, About once per year.
3. Over the length of the project, how much more work is discovered and added to the original estimates?
    a. Less than 10%, 10-30%, 30-60%, 60%-100%, greater than 100%
4. How is the discovered scope shared with project leadership?
    a. Respondents will type in their answer
5. How effective are the mechanisms for sharing the discovered scope with Project leadership?
    a. 7 levels from not effective to very effective
6. What is done to adjust the expectations of the project leadership to account for the newly discovered work?
    a. Respondents will type in their answer
7. How often does project management adjust project scope or delivery targets in response to the discovery that there is more work to do than originally estimated?
    a. Never
    a. About yearly
    b. About quarterly
    c. About every other month
    d. Monthly
    e. Per sprint
    f. Weekly
2. How effective are the project management activities with keeping the delivery targets aligned to the newly discovered work to be done?
    a. 7 levels from not effective to very effective

Appendix 2: Survey Results from the Software Developers

Question 1 (Single Answer) - Overall Results - 3 Respondents (extremely small sample size)

Question

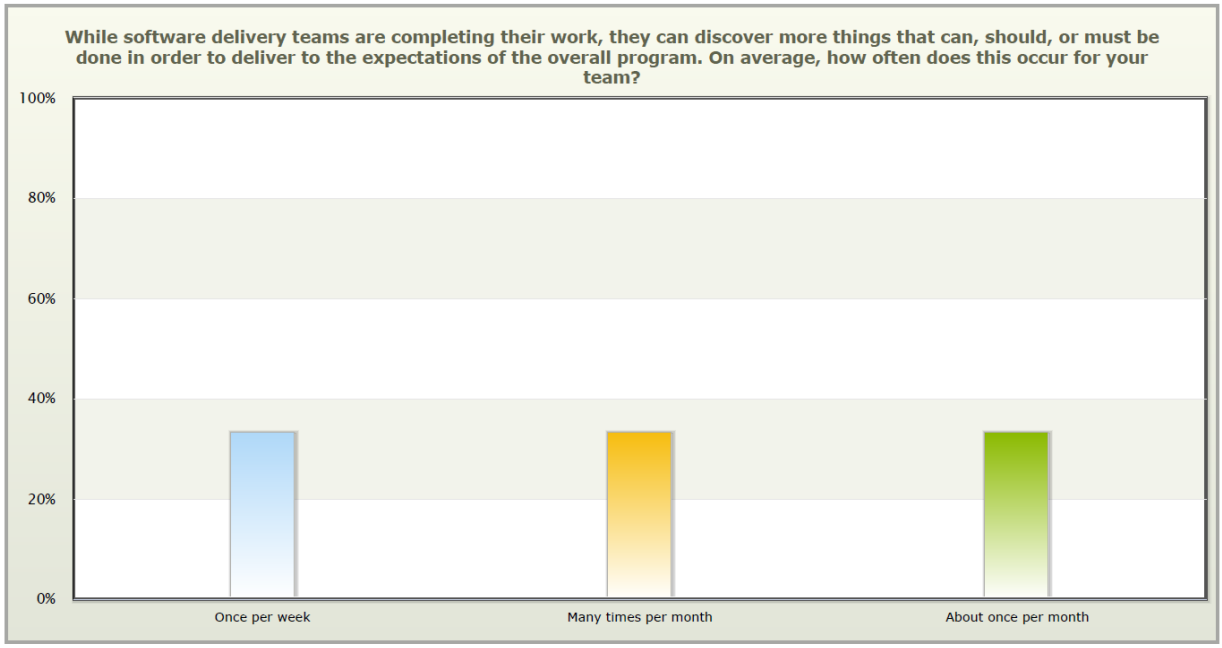
Overall, how effective are the project management methods of this organization with respect to managing software delivery?



Question 2 (Single Answer) - Overall Results - 3 Respondents (extremely small sample size)

Question

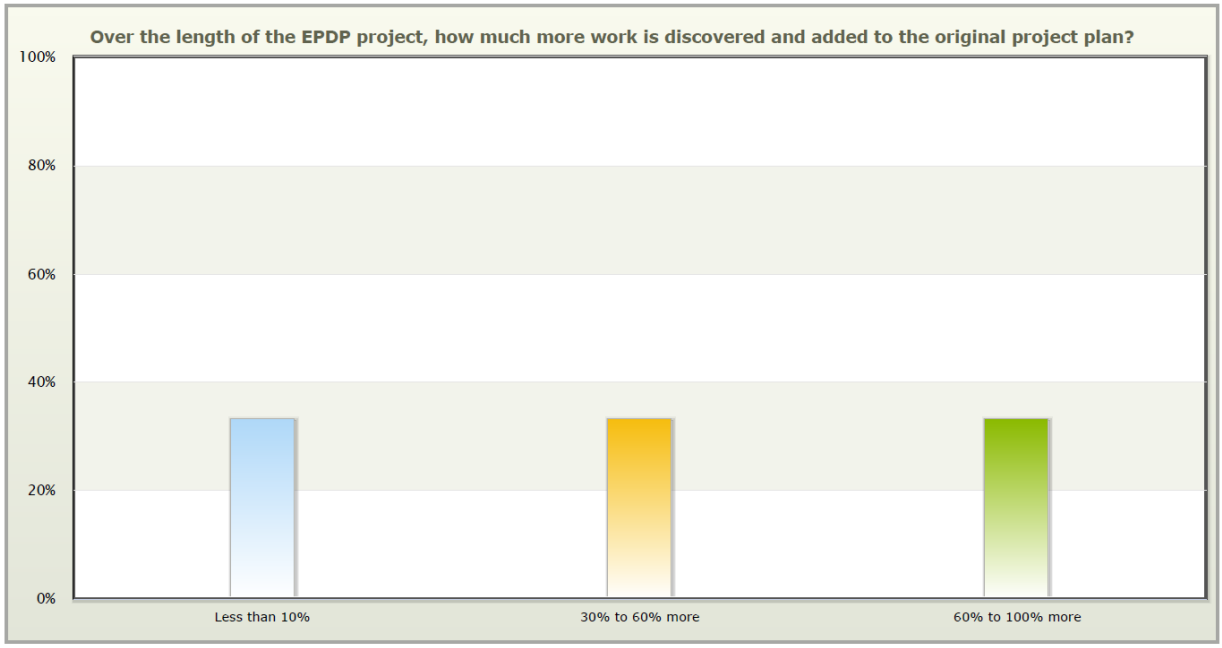
While software delivery teams are completing their work, they can discover more things that can, should, or must be done in order to deliver to the expectations of the overall program. On average, how often does this occur for your team?



Question 3 (Single Answer) - Overall Results - 3 Respondents (extremely small sample size)

Question

Over the length of the project, how much more work is discovered and added to the original project plan?



Question 4 (Single Answer) - Overall Results - 3Respondents (extremely small sample size)

Question

How is the discovered scope shared with the program leadership

Export List

Open Answers

Thru status meetings and aglie "epic"

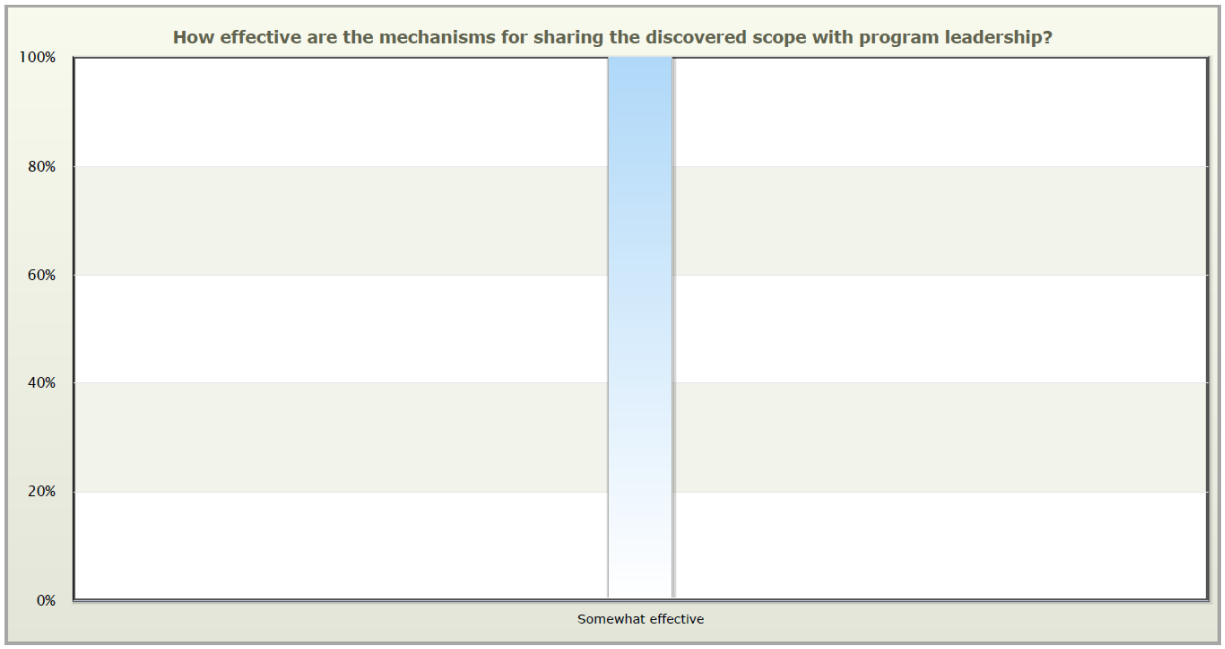
Via planning activities like Pre-Planning or PSI planning

Usually, this is shared through a formal meeting, i.e. PLT meeting

Question 5 (Single Answer) - Overall Results - 3 Respondents (extremely small sample size)

Question

How effective are the mechanisms for sharing the discovered scope with program leadership?



Question 6 (Single Answer) - Overall Results - 3Respondents (extremely small sample size)

Question

What is done to adjust the expectations of the program leadership to account for the newly discovered work?

Export List

Open Answers

Text | Word cloud (beta)

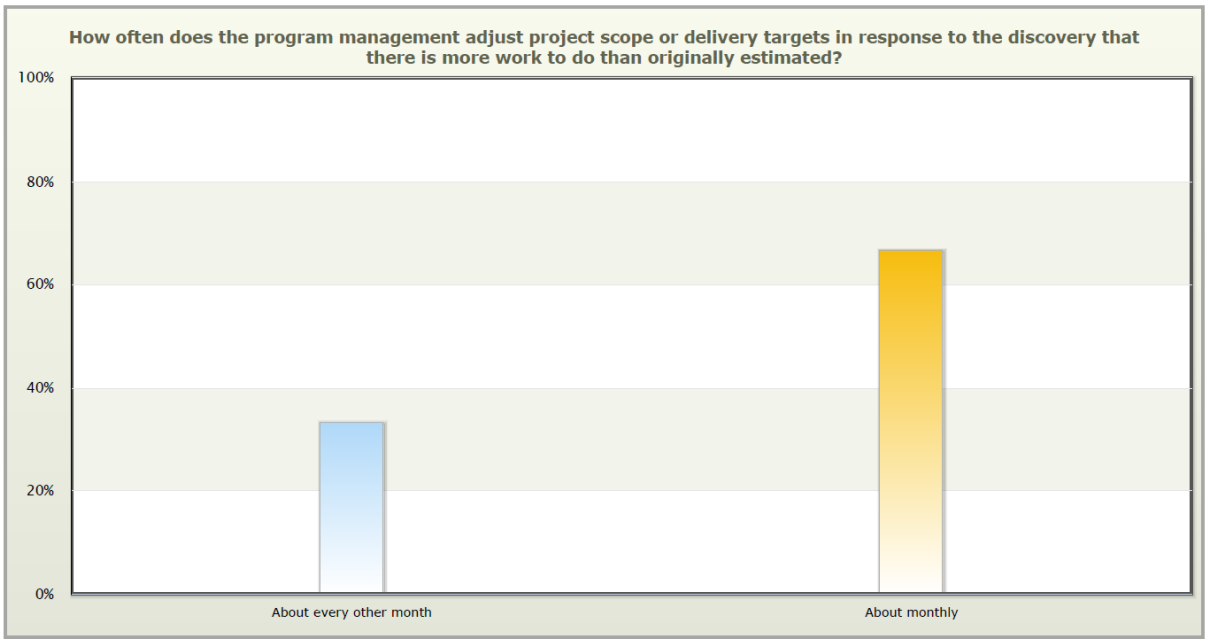
No extra time and no extra resources are given to pursue the newly discovered work.

Leadership doesn't seem surprised or have concerns about newly discovered work.

Question 7 (Single Answer) - Overall Results - 3 Respondents (extremely small sample size)

Question

How often does the program management adjust project scope or delivery targets in response to the discovery that there is more work to do than originally estimated?



Question 8 (Single Answer) - Overall Results - 3 Respondents (extremely small sample size)

Question

How effective are the program management activities with keeping the delivery targets aligned with the newly discovered work to be done?