Beyond Software: When Agile Meets Defense Systems and Hardware

WARREN B. SMITH, General Dynamics

As government and defense organizations embrace Agile Development, one of the stickiest issues is how to integrate systems engineering and embedded hardware design into an agile program. Defense systems often require significant analysis and long-lead times to produce special-purpose hardware with circuit cards and housings specially designed and assembled in-house along with components from multiple vendors. Because agile assumes a low-cost-of-change, these systems and hardware disciplines struggle to be agile in the face of significant analysis and long-lead times. This experience report describes how our team at General Dynamics launched an agile development project and adapted Systems Engineering and Hardware Design teams to this environment. The changes were widespread, from planning through implementation. This report discusses the importance of model-based engineering to take agile beyond software, and details how specific team structures can support agile systems and hardware projects.

1. INTRODUCTION

This experience report focuses on how our team at General Dynamics initially applied agile principles to systems engineering and hardware development (electrical and mechanical). We are now applying these lessons to other efforts.

General Dynamics' objectives for moving to agile practices are threefold: deliver highest-value items earlier; better predict completion times; and improve quality. Due to the nature of our systems, we must deploy agile beyond software to systems and hardware engineering to take advantage of these promised benefits.

A common objective in agile software development is producing working, tested, demonstrable software each sprint. How does that translate into a backlog for systems engineering? Or embedded hardware designs needing 6 months for parts procurement? In both areas, moving to agile methods meant changing our mindset.

Common agile tasks such as defining a backlog challenged our thinking in systems and hardware. For example, software engineers often write stories in a format ("As a *user type*, I want to *achieve a goal* for a *reason*") that does not clearly work for systems or hardware. Historically, our systems and hardware engineers often defined tasks based on lifecycle activities, such as "analysis," "design," and "layout". We evolved toward defining backlogs as either work products, or system functions. Shifting our mindset took time, a precursor to changing our behavior and becoming more agile. Our efforts finally took off once we shifted from activities to work products, structures and functions, and incrementally defined items of value for systems and hardware.

Agile efficiencies require a low cost of change. This is an inherent characteristic of software development where changes are made by the keystroke. Historically, systems and hardware require long periods to design, procure and assemble physical components and see the effect of changes. Model-based engineering (MBE) now changes the nature of systems and hardware development and reduces this cycle time. By generating designs in a model representing the system before committing to vast levels of engineering effort and expensive components, MBE enables a cost-to-change similar to a software-only effort.

MBE provides an environment where, like software, the model can be changed quickly. In model-based development, the model *is* the system, just as the code *is* the software product. Think of the model as "revision 0" of the system. The model enables rapid updates, differing views of the system and show our progress to the customer. The model easily shows the impact of changes, verifies, and validates the expected system operation.

Warren B. Smith; General Dynamics; warren.smith@gd-ms.com; wsmith@gatech.edu
Certified SAFe® 4 Product Owner Product Manager and Scrum Master

2. THE SITUATION

The team at General Dynamics knew we would uncover unforeseeable challenges by changing our development from a document-based waterfall lifecycle to model-based agile approach. Recognizing this, our Department of Defense customer took a risk, identifying our project as a 'path-finder', offering the freedom to explore the cultural implications of agile and MBE. The customer gave the project latitude to find workable approaches and acculturate the organization. This project was a good candidate for agile/MBE. It was low risk, very similar to the legacy system it is replacing, and, as a support equipment system, would not be used in a life-or-death situation. The risk paid off handsomely.

Our project is contractually required to hold Department of Defense (DoD) waterfall acquisition milestones: System Requirements Review (SRR); Preliminary Design Review (PDR) and others. Our funding was structured accordingly with initial funding for systems engineering. We therefore applied agile thinking within this framework, with the goal of eventually replacing milestone reviews with a true agile approach.

We started with a significant systems engineering effort to decompose the contractual requirements using a series of model-based analyses to derive the system behavior and resultant requirements for SRR. The systems team also defined the subsystems, and allocated the system behavior to those subsystems. From this allocation, we derived subsystem functional requirements and design. The derived functional requirements became the basis for software stories. The subsystem design became the basis for the hardware design. This project required functional decomposition and system design to define the hardware and software team's stories.

In this report, "system design" and "subsystem design" means developing the physical manifestation of the system. See the *Systems Engineering Handbook*, 4th ed., Wiley, section 4.4.1.2 Page 64 for definitions. The system and subsystem design defines the structure (the hardware, software, and processes) that make up the system. People also play a key role because operators perform many critical system functions.

We used scrum as our agile framework, and created a single team to develop these system analyses. The team consisted of: Ken, the Project Lead who also acted as a Product Owner; Kelly, a Subject Matter Expert (SME) who also acted as scrum master; Fred, a retired-military expert helped the team with his vast knowledge. Phil is a certified Systems Modeling Language (SysML) modeling expert who deployed the tools and trained the team. The bulk of the team was comprised of early-career, highly motivated and very talented engineers. I acted as agile and modeling coach, based on previous experiences in agile systems engineering.

The team faced a number of learning curves. For example, the team chose Jira as its kanban tool, which was not part of the corporate toolset. Ken bought 10 licenses for $10 to get the team started. Over the course of the next year, Jira's use has expanded dramatically beyond this team, demonstrating its value.

The team also faced a learning curve with the model-based engineering (MBE) tool suite and the modeling language, which was critical to reducing our cost of change and by extension, our agility. In addition to providing tool training, Phil deployed MagicDraw© as well as companion tools supporting integration with other tools, and on-line review-and-comment, which dramatically reduced our peer-review times

3. HOW WE STARTED

In 2017, the project planned to use a waterfall approach. By Feb 2018, our managers agreed to path-find agile and MBE. We therefore formed the first agile modeling team (a systems engineering team) in March 2018. Our goal was to complete our SRR in July, using model-based analysis to derive system and subsystem requirements using scrum, operating 2-week sprints, sprint planning, and daily stand-up meetings.

Influenced by agile software projects, our initial backlogs contained very short tasks. For example, if we needed to produce a system block definition diagram, we broke down the task into chunks of work that could be accomplished in ½-3 days, maximum. This was no small feat when defining systems engineering work!

We therefore settled on backlog item tasks that included items like "look-up legacy requirements", "locate and read interface definitions", "understand system operation from Fred", culminating with "draw diagram". The team performed many of these tasks in several hours, and was very successful at moving backlog items to "done". Defining these steps also provided a new, tremendous advantage: with Kelly and Ken clearly defining the backlog as the small steps needed, our newer team members were fully engaged and highly productive. The team's excitement and progress was palpable as these small ½-3 day tasks moved across the kanban to "done".

3.1 Enter Earned Value (EV)

We encountered the first major challenge almost immediately, in April 2018. Back in 2017, we had developed our earned value (EV) baselines using waterfall planning. Once we decided to deploy in an agile manner, our work started to diverge from the plan. Only a month after the start of work, and three months left to the SRR, the EV system was already showing we would miss our SRR by over a month.

Such a delay was unacceptable, and the program management, while supportive of agile, faced intense scrutiny from our Customer and did not yet have enough information to explain the EV variances. Pressure to add staff was immense. This diminished the confidence in our approach: would our nascent agile effort be terminated before it could bear fruit?

On the other hand, the engineering staff, with a fully groomed backlog including *all* tasks to SRR and a clear measure of progress, was supremely confident we would meet the deadline. Faced with the EV data, Management did not know if this confidence was warranted, or if Engineering had just gone mad. Agile had exposed a gap between the engineering team's view of the project and the EV data.

To support engineering, I performed a points-to-hours conversion, determining the number of hours we had expended for the points produced and calculated an hours-based velocity. Projecting this forward against our backlog, predicted that we were in fact on schedule. Using this data, engineering and program management got on the same page, and we were able to provide rationale for the Customer to trust our approach. Kudos go to Lindsay, the Program Manager, as well as the Customer, for supporting the effort through this period.

We concluded that the reason why there was a discrepancy between the agile projection and the EV projection was twofold. First, our productivity increased using agile methods. Second, executing an agile approach using a waterfall EV baseline was clearly incompatible.

Lesson One: Agile Systems Engineering provides more accurate predictions for near-term work.

3.2 Adventures in cross-functional teams

To improve the quality of the analyses and backlog grooming, we added software engineers to the agile systems team. Software engineering had experience with agile development. By working together, we increased the quality of our analyses for the software teams, who would use these analyses as a basis for their stories. In late May 2018, Mike, a visionary hardware manager, approached me to also integrate hardware onto the team. While we decided to pursue forming a hardware team after the SRR, we immediately embedded Josh, the Hardware Lead onto the systems team, just as we had with the software engineers. The integrated team collaborated better and kept hardware's needs in focus. This close collaboration of systems, software and hardware engineering decomposing the system functions and structure worked very well.

By now you are probably thinking, "What is going on here? Agile in a waterfall project? A systems engineering team? This has nothing to do with agile!"

Other than this natural cross-pollination of hardware and software membership on the systems team, it was not obvious what "cross-functional" meant when expanding agile beyond software. Agile software teams consider themselves cross-functional if they have all the people they require, such as software developers, software testers, and a UX member. This does not appear cross-functional at all to a systems engineer. To systems engineers, a cross-functional team includes systems engineers, hardware and software engineers, test engineers, maintainability, reliability, safety, production, human factors and a host of others. Is there a "cross-functional" role for a hardware engineer on a software team, or vice versa? The nature of their work is so different; it certainly was not obvious that there should be. In contrast to the true cross-functional system/software/hardware team described above, we concluded that hardware engineers working on software design teams or vice versa did not necessarily make sense. Coordination is critical, but the nature of the work itself is very different. Based on this experience, we now have a good understanding of where and how to structure cross-functional systems and hardware teams. In fact, section 7, discusses our current thinking in this area, including embedding a systems engineer as a named role on every team.

Lesson Two: Complex backlog functional decomposition and system design may require systems engineers.

Lesson Three: Teams developing either software or hardware can be functionally cohesive.

4. SYSTEM REQUIREMENTS REVIEW

Modeling changed the nature of the work from March 2018 through SRR. Instead of a document-centric approach, the team's model-centric analyses defined system functionality and developed structural diagrams. The system functions were decomposed in the model and allocated to the various subsystems.

In early July 2018, we held a very successful SRR on time. In previous projects, teams spent significant time creating PowerPoint decks to present at reviews. The teams were not sure how much value the PowerPoint offered for the time spent producing them. Instead, we chose to present the work itself, presenting a "live" walkthrough of the model during the review. Explaining the models provided more insight into the system than any review of textual requirements had in the past, delighting our customer.

This successful event had the secondary effect of validating that agile provided a more accurate near-term assessment of productivity than the EV data had (at least in this case). The customer was very happy with how our agile development was progressing. Following SRR, Ken created new EV work packages mapped to the backlog items for upcoming sprints leading to PDR. This has generally aligned our EV to our actual work productivity, greatly reducing our EV variances. We started a software team, planned our hardware team, and brought in Cindy as the scrum-of-scrum master.

Lesson Four: Agile requires a balance when defining EV packages.
Lesson Five: This customer appreciates reviewing live work products.

5. GROWING THE PROJECTS (DEPENDENCIES, SCOPE CREEP, AND METRICS)

5.1 Dependency Planning

After SRR we added more teams. The software team was in full swing. Another MBE team joined to provide Simulink models to the effort. Hardware, while not yet formed as an agile team, actively participated. Finally, quality assurance individuals began supporting the effort.

This growth in participants added dependency risk. After collaborating with Max, an experienced agile coach, I facilitated the first in a series of quarterly planning meetings to coordinate among these individuals and teams. Everyone participated in working meetings to identify the inter-team dependencies. I invited key hardware people to get acculturated to agile. Ken presented the objectives for the next 3 months, followed by a round-robin discussion where each team (and support individuals) identified their dependencies on other teams. The teams groomed their backlogs and worked together until all dependencies were identified. Cindy created a "dependency board" in Jira which captured each dependency, the work product, who produces and uses it, and due dates. Cindy took over leadership of the second such event, and has run it with efficiency ever since.

The value of this event was immediately obvious. The considerable number and details of interdependencies between the teams had previously been poorly understood. Before the dependency planning, work products critical to one team were not necessarily on the supporting team's radar. One key area was procurement of "long-lead items" or LLI. These are hardware parts that are time consuming to design and procure. Sometimes these parts cannot be ordered until trade studies, requirements analysis or design has been finished. While no parts were ordered late on this project, the hardware team was always uneasy that the systems and hardware trades were done on time. Adding to this concern, the contractual obligation to do up-front systems analyses made it more difficult to prioritize the hardware efforts based on procurement. These conditions highlighted the need to kick off an agile hardware team.

5.2 Metrics and Scope Creep

By September 2018, Ken was required to show metrics every month to engineering management. In addition to EV, we used metrics generated from Jira showing epic burndown, velocity and version reports. These were new to the organization unfamiliar with "Epics", "Velocity", and the meaning of points. We pushed to identify a standard set of metrics. Based on the increasing visibility of the project, I was prompted to organize a series of metric summits with several of our company's coaches. Ken, Kelly, Cindy and I hashed out metrics with Max (our first agile coach) and Rob, an expert agile metrics coach. Their insights into our backlog were incredibly helpful. They sat through hours of working sessions with the team, providing invaluable aid into the metrics, and how they measured the team's productivity. While agile had made the team much more productive. Max and Rob's insights were critical in further improving our team's performance, and the impact was far-reaching.

Our examination of metrics exposed our scope creep. The team was in the habit of adding stories whenever we identified new tasks contributing to a work product. While our task-based backlog worked well for ensuring our less-experienced staff was highly productive, it became clear that we were adding newly pointed *tasks* needed to produce a work product, without adding scope to the work product *itself*. The team added points to account for extra activities. For example, when working on an analysis, we would add *tasks* when coordinating with additional people without changing the overall *value* of the final analysis.

We realized we had an activity-driven, not product-driven backlog. While each task on the backlog was *necessary* to produce an item of value, we were not tracking the items of value *themselves*. It was our first understanding of the power of metrics. This forced a discussion as to what an "item of value" is in the systems engineering domain. In agile software, a tested, demonstrable software module is the standard "item of value".

In Systems, our “end user” is just as likely to be an internal engineering department who needs our work product, as an actual user or customer. An item of value tended to be a system structure, a system function or a work product required by someone else.

We therefore decided to completely re-build our backlog. Gone were the old activity-based tasks, replaced with those work products that provide value to the next stage of development. Is an interconnect model required to define cables? Then the interconnect diagram *itself* is identified as a story. We captured the previous activities in the description, and slavishly captured definition of done and acceptance criteria. We also replaced our small ½ day tasks with longer ones, representing valuable work products, rather than activities. This worked well as most of our systems work simply is not completed in a matter of hours or days.

The impact was immediate and dramatic. Our scope creep ended overnight. Every item on our backlog was needed by either an end-user, the customer, or the next stage of development. Sadly, also gone was our sense of rapid progress, as getting items over the goal-line, into “Done” on the kanban took much more time.

The team also struggled with our new increase in “splash,” work we did not finish in one sprint that we moved into the next. Some of the larger work-products of value took longer than one, 2-week sprint to complete. Ken, Cindy, Max and I discussed this, assessing whether we should lengthen systems sprints to 3 weeks, attempt to define smaller work products (again), or accept the splash. Ultimately, Ken, as the product owner and project lead, decided it was better to have the flexibility to change direction every 2 weeks than try to further reduce splash. Since some work-products were produced in a 2-week sprint, others in 4 weeks, Ken valued the flexibility 2-week sprints offered to react to changes in priorities. This decision offered predictability to within a 4-week period, which was adequate, plus flexibility to quickly change direction. In short, the metrics had done their job. They had exposed that our planning was task-driven, rather than value-driven, and we adjusted accordingly. We standardized on a set of metrics, presenting the version report to management showing our progress towards hitting the next important milestone. We now include versions in Jira for all upcoming milestones for those backlog items. Teams also use Cumulative Flow and Control Charts.

Lesson Six: Some systems work products require longer sprints (3-4 week) to produce (without splash).

Lesson Seven: Stories in the systems domain often represent deliveries to the next development stage, rarely directly to the users themselves.

5.3 Key Systems Handoffs to Hardware and Software

Before MBE, the primary hand-off from systems-to-software or systems-to-hardware was a set of textual requirements and design descriptions, consisting of various block diagrams and other analyses. Each team used different formats for their work: Software teams would create ARTiSAN® models. Hardware teams would create layout and interconnect diagrams in Visio or CAD tools. The model-centric approach eliminated those hand-off inefficiencies. With MBE, the software teams *refined* the system models by further decomposing them. Similarly, hardware teams *refined* MagicDraw interconnection diagrams created by systems with more details for cable definitions, etc. Refining the system model with hardware and software details boosted productivity.

At SRR, software published a release schedule. Our process requires systems to verify software releases against the requirements. Consequently, concurrent with the dependencies, I socialized systems analysis in one sprint feeding software development in the next. While the software team develops, the systems team writes verification procedures, which are used to test the software once it is completed.

Lesson Eight: Including systems engineers on cross-functional software teams can support functional decomposition and verification for that team.

5.4 Forming the Hardware Team

Designing embedded defense-system hardware is a time-consuming process. We custom design and build because the systems need to operate in difficult environments such as temperature extremes, salt-fog near oceans, hoar-frost or steamy, fungal-friendly jungles. The system might be susceptible to acute vibration or physical shock. Those environments can destroy most hardware. It takes custom components, long-lead parts, and expensive “qualification testing” to design hardware to withstand these conditions.

We had already co-located Josh, the hardware lead with the systems team. This helped by providing closer coordination, visibility and feedback with the systems activities. Helping the hardware team switch to an agile approach took longer than we expected. The hardware design culture, well suited for waterfall, was the primary hurdle. Specifically, the hardware design teams defined long-term activities in development phases, such as “designing”, “layouts”, etc., which was not well suited to conversion to a backlog based on ‘things of value’. Their plan represented activities, instead of products.

Unrelated to agile, Mike was coaching his team to focus on deliverables, using a "punch list," for planning. This work-product-oriented list was just the approach we needed for grooming a hardware backlog. Josh abandoned the activity planning. Familiar with how the Systems team operated, he expanded on the punch list and built an initial backlog for use by a new hardware team.

Launch. Once Josh had a draft of the hardware backlog, we held a meeting with the broader group of hardware managers to introduce them to the basics of agile development. At that meeting, we explained how the hardware team would operate on the project. The hardware organization had heard of agile, assuming it was a software-only development process. Many people had little understanding of any details regarding agile.

At the meeting, I described the production origins of lean techniques and the development of kanban on production lines. Ken discussed pointing. We identified the objective of defining smaller value-added work products and tracking work through workflow states on a Jira kanban board. Josh then went through the backlog. The initial response was mixed. Some were enthusiastic, and a few people pushed-back on the approach. The meeting outcome required Josh to perform additional backlog grooming and planning, and I performed further coaching with the hardware managers.

Team workflow and cadence. The hardware team defined the workflow for their kanban board, which itself was challenging. Significant portions of the hardware organization were outside the control of the project team. Specifically, the drafting department was set up as a service organization and somewhat isolated from the project: the project submitted design information to the drafting group, which assigned drafters based on rotating priorities. Therefore, the team did not have dedicated drafters. In response, we added "Drafting" to the workflow to accommodate this work phase, and captured the bottleneck created by this uncontrolled work state. Other service organizations (like procurement) received similar workflow states. With the backlog and workflow in place, we started the hardware team. This highlighted that converting hardware to agile requires addressing broader service-based organizations that span many projects.

The hardware team selected 3-week sprints to accommodate the longer times needed to produce work products. Even so, splash continued to be an issue. The "Drafting" workflow state drove splash, which the team was unable to control. Splash also resulted when team members were reassigned to other short-term, high-priority tasks unrelated to our project. The service-based mentality also lends itself to a willingness to pull people off teams to work on other high-priority tasks. This was not the only example of this phenomenon: the team was often at the mercy of procurement and other department's differing priorities. The team adjusted by reducing their capacity to accommodate this anticipated time loss from its own members. This work-around was effective but undesirable: it affected the team's progress, and incurred productivity loss due to context switching. This situation highlights a general challenge to transforming an organization to agile development. Every agile team operates within constraints. Our hardware teams (and sometimes other teams too) had to work within a "shared service" organization. That meant our agile teams needed to fight for dedicated people or somehow accommodate the bottlenecks imposed by the non-agile portions of our organization.

Lesson Nine: Shared services do not work for agile teams. Invest the time to get specific people assigned to your agile team, even if the person can only work part-time for your agile team.

Successes. Irrespective of these hurdles, the hardware team made tremendous progress on a number of fronts. First, the team defined work products that add value in a short amount of time. The hardware work products included defining system interconnections, creating or updating schematics, creating cabinet block diagrams, selecting connectors, laying out cables, and creating specific drawings. The team broke out most work into value-added chunks that could be performed in about a week.

The hardware teams also successfully prioritized long-lead parts earlier. It is important for hardware design projects to identify the long-procurement items, design them first, and get those parts on order. The granularity afforded by the hardware backlog made this more precise. Since the systems team needed to perform sufficient system design analysis to feed requirements to the hardware team, prioritized by long-lead parts, this benefit brought to light another key lesson of agile hardware development:

Lesson Ten: Working together, systems and hardware engineering can learn to define system design increments (or "slices") faster, reducing long lead times.

Applying a lesson learned by a long-term colleague and friend, Jozsef Bedocs, with whom I have collaborated on other projects and papers relating to this topic, as well as insights from Max, yielded a corollary to lesson ten. Often the software teams need specific hardware structures in place in a timely fashion to support their software story deployment.

Lesson Eleven: In addition to analyzing system behavior for implementation by software teams, Systems efforts should include "system design analysis" to ensure hardware is in place to support software deployment.

The team's third success was integrating the hardware and systems teams using MBE. Traditionally, the systems team identified interfaces in one tool and the hardware team defined the interconnections in another.

This project did things differently. The systems team identified the interfaces in the model, collaborating with the hardware team. The hardware team then *evolved* those interfaces by refining them into detailed interconnections, in the same model. This evolutionary approach leveraged the model's consistency checking, error trapping, and other features to greatly reduce defects. This activity is identical to the corresponding software team refining system block and behavior diagrams to more detailed software design.

As work proceeded, the hardware team, led by Josh, presented their model-based and agile efforts to hardware management, with Mike's and my support. There was much dialog between management and the team, and keen interest in the approach and results. Hardware management was very happy with the results of the actual work accomplished! This contrasted to the first meeting, where the same group was trying to visualize the theoretical, promised results. To say this meeting was incredibly successful is an understatement: it was the first time I have seen such a technical presentation garner applause.

6. PDR - EPIC-BASED DEVELOPMENT

We prepared for our PDR (Preliminary design review) by continuing our shift away from waterfall. Unlike waterfall, which reviews the entire design, we were now completing threads of functionality and slices of hardware structure. At PDR, we reviewed our completed work.

This more agile approach resulted in a very different PDR. Instead of the traditional review of all materials at one meeting, the team instituted a phased-delivery, organized by topic. The topics included software analysis and design of certain groups of functionality. The hardware design was also one of the topics. Our teams published each topic incrementally from the model for customer review. The customer provided technical comments against each topic incrementally, instead of reviewing it all together at one PDR meeting. This process provided the customer with a much more thorough assessment of the material. To our surprise, the added quality came with added cost as comment adjudication took longer than anticipated. On the other hand, ever since this project, our use of PowerPoint to present work products to the customer, and the associated costs, has dwindled dramatically, in favor of presentation of actual work products.

Lesson Twelve: Our need for external review has not changed. By using actual work products, we still accommodate external review, while spending less time on decks and more time on product development.

7. MOVING FORWARD

This path-finding project has taught us key lessons that are informing all our new agile efforts. Working closely with Max, who is independently bringing additional and complementary agile concepts to systems engineering, we are now organizing teams differently to apply these lessons to new projects.

7.1 Grand Agile System Design, or "Architecture," "Function," and "Design" Teams

Through our experiences, we realized that projects would require a new team responsible for the "System Design". This team's composition and role will differ from agile software teams deploying stories.

Architecture Team: Defining system structure. Based on our path-finder project, we are now implementing the following structure on a new program. We have a system design team, called an "architecture team" responsible for defining the overall system *structure* needed to host *functionality* deployed by function teams, using MBE to capture the results. "Function teams" refer to agile software teams implementing stories to deploy new system functions.

Prioritizing and synchronizing system design. The architecture team prioritizes the embedded hardware design work needed for the overall system design. This true cross-functional team is comprised of systems, hardware and software architects, and prioritizes hardware design by "slices," or iterations. The team specifies each slice using models, scheduled to support the function team's software deployment schedule.

Deployment strategy. Finally, the architecture team defines a *deployment strategy* to ensure each hardware iteration is available on time. The deployment strategy might deploy breadboards to support earlier software deployment and test, followed later by prototype, then production hardware. The architecture team prioritizes each hardware iteration *and* strategy to ensure structures are available in time to host function team's stories. For example, let us assume the architecture team needs to deploy a microwave communications infrastructure. This team would specify the structures needed, and in what order. The first iteration may be a wired data communication backbone to wring out the applications interfaces. Function teams could test their software on real hardware early. At a later iteration, the architecture team would replace the wire with prototype line-of-sight microwave communications. Finally deploying environmentally-qualified hardware.

Design Teams: The architecture team passes a specification for each slice to a "Design team," which iteratively refine the models to implement each structural slice. Design teams are largely hardware-focused, iteratively implementing the specifications to design each slice to ensure its fastest possible deployment.

Function teams often have 2-week sprints, architecture and design teams may operate on 4-week sprints.

The benefits of this structure are three-fold. First, by implementing the system design in "slices" of related hardware infrastructure, it replaces the defense hardware phase-gate approach with an iterative approach.

Second, the architecture team prioritizes hardware structure to be in-sync with the function (story) deployment. They also specify and prioritize each slice needed to accommodate LLI procurement.

Finally, the architecture team defines the hardware deployment strategy, deciding if each slice is initially deployed as a model, breadboard, prototype, or qualified hardware

7.2 Embedding Systems Engineering on all Teams

In our complex defense systems, backlog decomposition can require significant systems analysis in and of itself. This typically is the domain of systems engineers, who have tools and skillsets for decomposing behavior to stories. Max and Jozsef, through their independent work, have hit upon the idea of adding a systems engineer to each team. So, in addition to forming the three types of teams above, our new project is embedding a Systems Engineer on each team alongside the product owner and scrum master, to perform the functional decomposition and other systems analyses. These systems engineers collaborate across teams and with product owners to ensure story decomposition is complete and consistent. These systems engineers tend to operate "in advance" of the rest of the team, defining the parameters for stories, which will be implemented 1 or 2 sprints later.

8. RETROSPECTIVE

General Dynamics' large-scale agile expedition beyond software has been fruitful, and the lessons learned are already accelerating our transition to a full agile deployment for our Defense Systems.

Our path-finder project applied agile to a waterfall framework. While it played a critical role in shifting the culture, many people outside the project felt we missed an opportunity by not implementing a "big-bang" agile implementation. The path-finder project has brought to light very significant challenges that pragmatic Agilists must face when seeking to extend agile beyond software. Treating agile with religious fervor does little to overcome the very real constraints faced by organizations. We use the term "cultural change" lightly, sometimes flippantly. Changing the many facets of culture (engineering, management, finance and customers) impacts behavioral norms, mindsets, organizations, and budgets. And of course, change is risky.

It has been an honor to work with this talented path-finder team. These people broke new ground and accomplished so much! They worked, socialized, faced and resolved conflicts together. High functioning teams, made up of compatible people, are the backbone of successful organizational-wide agile deployment.

9. ACKNOWLEDGEMENTS

General Dynamics' policy prohibits naming individuals in public papers, so first names (fictional or not) have to suffice. Pioneers take all the arrows, and Ken, the project leader took many risks with agile and MBE. His leadership has been instrumental to the project's success. Fred was a key technical mentor, and Kelly was the technical tour-de-force behind our project's model. Phil, who deployed MagicDraw and patiently coached the team, built the infrastructure practically single-handedly. Cindy's work coordinating the scrums kept our disparate teams in sync. Josh fearlessly leapt into the unknown by taking leadership of the first hardware team and backlog. Mike, Josh's manager, was a visionary in "agile hardware". Max, my partner-in-crime in thinking through the new forms of agile for systems, is standing up the architecture team on our new project. Rob, our agile coach and metrics expert, drove many hours to support our efforts. Lindsay, our program manager, tirelessly defended the project. Many contributed in important ways including Matt, Kent, Lindsey, Elyssa, Ben, Roger, Justin and others. Jozsef Bedocs is a consultant who, while not working on this project, is my friend, my collaborator on other agile projects and papers, and frequently influenced my thinking. Brian, our manager on this project provided consistent guidance and support to the team. I'd like to thank my Shepherd, Johanna Rothman, for her very valuable insights, her enormous help with my writing style, and for bridging the defense systems and hardware world, with the agile software community. Finally, I'd like to thank Wendy, my wife and proofreader, for her support during the late nights and long weekends spent writing.