To be presented at 26th Annual INCOSE International Symposium (IS2016)
Edinburgh, July 18-21, 2016

The Best of Both Worlds: Agile Development Meets Product Line Engineering at Lockheed Martin

Susan P. Gregg, Rick Scharadin
Lockheed Martin
{susan.p.gregg, richard.w.scharadin}@lmco.com

Paul C. Clements
BigLever Software
pclements@biglever.com

Abstract. Agile development has long been touted as way to optimize software development team efficiency and improve project success. Product line engineering (PLE) brings large-scale improvements in cost, time to market, product quality, and more. Can these two paradigms work in concert with each other? This paper details the experience of Lockheed Martin as it introduced large-scale agile development practices on one of its largest and most successful product line engineering efforts.

Introduction

Agile software development refers to a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change [8]. Its adherents tout higher quality systems, delivered faster, which much better match customer needs and expectations.

Systems and software product line engineering, or "product line engineering (PLE)" for short, is a way to engineer a portfolio of related products in an efficient manner, taking full advantage of the products' similarities while respecting and managing their differences. Considering a portfolio as a single entity to be managed, as opposed to a multitude of separate products to be managed, brings enormous efficiencies in production and maintenance; these efficiencies are delivering order-of-magnitude improvements in engineering cost, time to market, staff productivity, product line scalability, and quality [9].

What happens when an organization tries to apply both of these groundbreaking, organization-changing methodologies at the same time? Can they work together at all? Is PLE, which relies on cross-product planning and well-entrenched coordination, compatible with Agile, the very essence of which is exceedingly short feedback loops and the ability to pivot as needs change?

This paper conveys the experience of Lockheed Martin, the world's largest defense contractor, as it is applying PLE and Agile together on one of its largest and most important projects. Not only is the project highly visible with demanding requirements, it is also very large, comprising some 10 million lines of code. This setting would challenge either methodology by itself; putting both of them together is yielding many lessons. At the end of the day, however, the answer to whether the two methodologies can work in harmony is resoundingly affirmative.

This paper will show how Lockheed Martin is using the two methodologies to support each other, and the lessons it is learning as it does so.

What Is Product Line Engineering?

Product line engineering (PLE) is a way to engineer a portfolio of related products in an efficient manner, taking full advantage of the products' similarities while respecting and managing their differences. By "engineer," we mean all of the activities involved in planning, producing, delivering, deploying, sustaining, and retiring products.

Born in the 1980s in the software field, but now having grown well beyond those early roots, PLE offers large savings observed from engineering the whole family rather than separately engineering each member. Numerous case studies [1][4][5][10][14][15] show that exploiting the commonality throughout the products' total life cycles can return substantial improvements in time to market, cost, portfolio scalability, engineer productivity, and product quality [13]; no other engineering paradigm shift has, to our knowledge, brought about results that rival these.

PLE as a Factory

An analogy with factory-based manufacturing serves to illuminate the important concepts.

Manufacturers have long used engineering techniques to create a product line of similar products using a common factory that assembles and configures parts to produce the varying products in the product line. For example, automotive manufacturers can create thousands of unique variations of one car model using a single pool of parts carefully designed to be configurable and factories specifically designed to configure and assemble those parts.

In PLE, the configurator is the factory's automation component; the "parts" are the assets in the factory's supply chain. A statement of the properties desired in the end product tells the configurator how to configure the assets.

Figure 1 illustrates. This factory's supply chain is at the left, in the form of shared assets that are configurable because they include variation points that are expressed in terms of the features [8] available in each of the products. A product specification at the top tells the configurator how to configure the assets coming in from the left. The resulting products, assembled from the configured assets, emerge on the right. This enables the rapid production of any variant of any of the assets for any of the products in the portfolio. Once this production line capability is established, products are instantiated – derived from the shared assets – rather than manually created.

The products in the portfolio are described by the properties they have in common with each other and the variations that set them apart. The products can comprise any combination of software, systems in which software runs, or non-software systems that have software-representable artifacts (such as requirements, engineering models, or development plans) associated with the engineering process that produces them.

In this context "product" means not only the primary entity being built and delivered, but also all of the artifacts that are produced along with it. Some of these support the engineering process (such as requirements, project plans, design modes, and test cases), while others are delivered alongside the thing being built (such as user manuals, shipping labels, and parts lists). These artifacts are the product line's assets.

Assets can be whatever artifacts are representable digitally and either constitute part of a product or support the engineering process to create a product. Four kinds of shared assets are shown in Figure 1, but those are just examples. Shared assets can include, but are not limited to, requirements, design specifications, design models, source code, build files, test plans and test cases, user documentation, repair manuals and installation guides, project budgets, schedules, and work plans, product calibration and configuration files, data models, parts lists, and more. Assets in PLE are engineered to be shared across the product line.

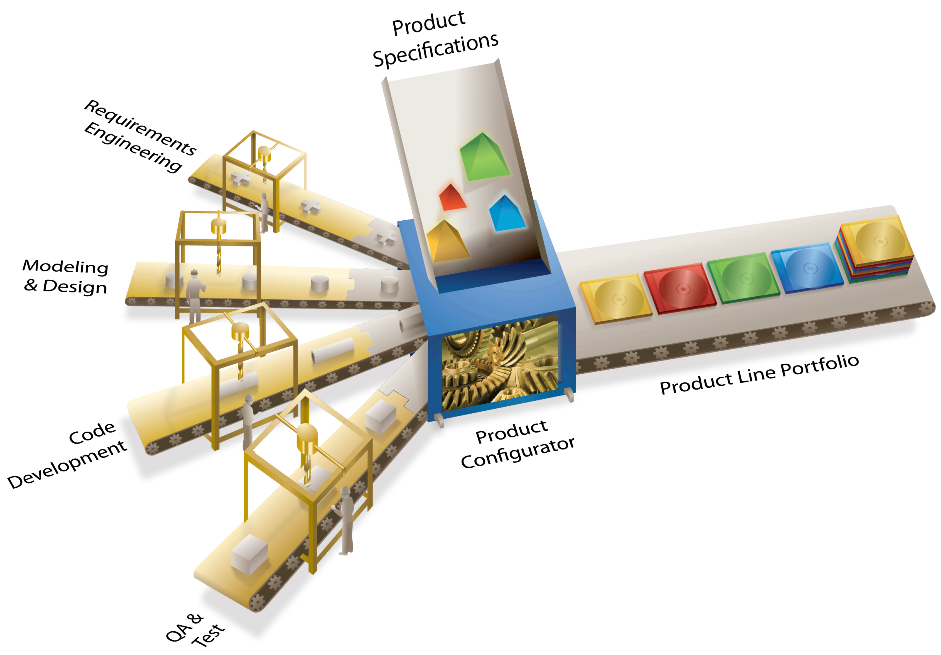


Figure 1: PLE as a factory.

PLE Contrasted With Product-Centric Development

PLE stands in contrast to traditional product-centric development, in which each individual product is developed and evolved independently from other products, or (at best) starts out as a cloned copy of a similar product that is then changed to suit the new product's specific needs. Product-centric development takes very little advantage of the commonalities among products in a portfolio after the initial clone operation. In particular, it derives very little benefit from commonality in a product's sustainment or maintenance phase, where data shows most products consume up to 90% of their project resources.

Figure 2 shows a production shop in which *N* products are developed and maintained. In this stylized view, each product comprises requirements, design models, source code, and test cases. Each engineer in this shop works primarily on a single product. When a new product is launched, its project copies the most similar assets it can find, and starts adapting them to meet the new product's needs.

To see how this form of reuse can lead to intractable complexity, assume that a defect is found in Product B, and that the defect is traced to an ambiguous or incorrect requirement in Product B's requirements. The Product B team fixes the error, re-designs as necessary, then fixes the code and test cases before re-deploying Product B. Product B is now healthy again.

But suppose that the defect in Product B's requirements was "inherited" when the Product B team copied the requirements from Product A. Suppose further that the source code for Product N was copied from Product B's (defective) source code, and the test cases for Product *N* were similarly "borrowed" from Product A's (inadequate) test cases.

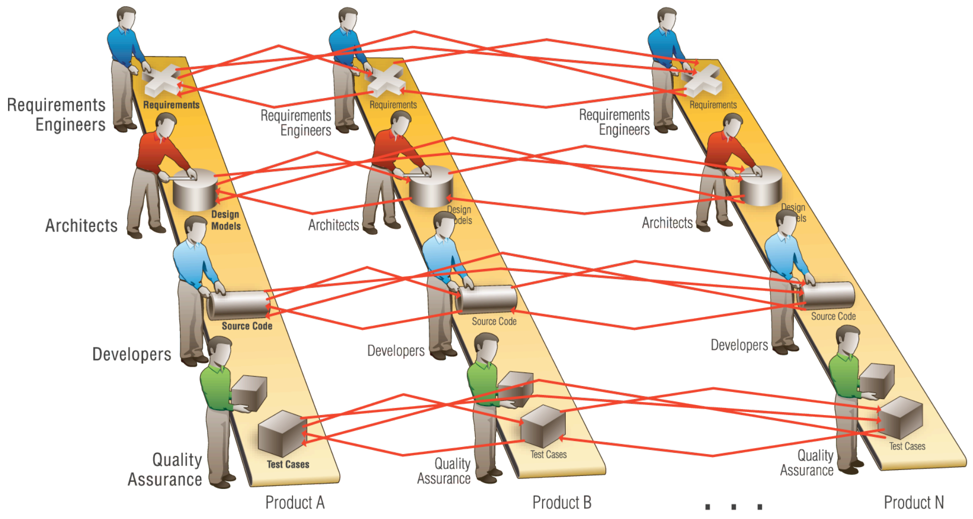


Figure 2: Product-centric development yields O(N^2) complexity

To really root out the defect from the entire portfolio, each of the *N* product teams should really confer with each of the other *N*-1 product teams. These communication paths are shown in red in Figure 2. This communication obligation imposes an overhead that grows as the square of the number of products. This complexity will quickly overwhelm any engineering staff; in order to get their products out the door on time and on budget, each product team will focus more on their product silo and less on taking advantage of the commonalities and interdependencies among the other products. The result is divergent product silos, low degrees of sharing, and high duplication of effort across the product silos to fix the same defect multiple times in multiple products, or to independently implement the same enhancements in different ways in different products.

Figure 1 alluded to PLE as a factory, and that analogy can be brought to bear to remedy the O(N^2) problem of portfolio management. In a manufacturing factory, a defective product would not be fixed by one-off repairs to the product itself. Rather, the factory, its supply chain, and the manufacturing process itself would be scoured to find the source of the defect.

So it is with PLE. Rather than fix a defective product, PLE engineers fix the shared asset(s) that need to be modified (perhaps by adding a new variation point) in order to produce the product correctly. Then, the configurator is used to re-generate the product, as well as any other product affected by the changes in the shared assets. Since re-generation has a low and fixed cost, it matters very little whether 2 or 200 or 2000 products need to be re-generated. Thus, fixing a defect, making a systematic enhancement, or carrying out any other kind of portfolio-wide change becomes an O(N) operation.

Product Line Engineering at Lockheed Martin

A product line of product lines

Lockheed Martin has for some years now been developing one of its flagship products for the US Government as a product line, using the PLE factory paradigm described earlier. By now the PLE processes and teams are mature and experienced, and view their primary objective as developing once, and building and deploying many times from one set of common assets – principally requirements, source code, and tests. Feature-based variation in requirements and code enables building a member of the product line with or without a specific capability.

First, some terminology that has emerged from their PLE approach:

- Lockheed Martin calls a member of its product line a *configuration*. A configuration might be, for example, an instance of the system to be deployed to a specific site. The product line approach, then, exists to produce configurations for customers. This product line we are writing about comprises roughly 100 such configurations.
- Each configuration that Lockheed Martin deploys to its final operational comprises nine major software components, working together to achieve the functionality of the full system. These software components are, in the context of their development projects, are called *products*[1]. The variation in the systems, then, manifests itself as variation within each of the products.

Each product is developed as a product line – a mini-"factory" – in its own right. Accordingly, each product has a team associated with it, and is managed as a project, with budgets and development schedules that must play into the budgets and schedules of the systems at large. Thus, the overall product line is actually a product line of product lines. A feature profile for the entire system is actually a set of feature profiles, each describing a desired configuration of one of the products (components).

The coarsest-grained variation point is to include or exclude an entire component depending on whether or not the feature(s) provided by that component are included in a configuration or not. If a component *is* included, it can be further varied by exercising variation points inside, again based on feature choices.

For requirements, the product line employs a common specification repository (a DOORS database) that contains all requirements for all configurations, with varying requirements captured in feature-based variation points. This model allows for multiple configurations to share requirements while having the flexibility for each configuration to have unique requirements as well.

During the test and verification phase, the product line utilizes a consolidated testing approach to maximize efficiency of common requirements and capabilities. This results in tailored regression testing based on changed functional area. An integrated test team also uses common test plans and procedures. Common test efforts are leveraged and consolidated problem reporting avoids duplicate reporting caused by redundant testing – all part of the PLE factory approach.

[1] This is a different definition of "product" than appeared in the introductory section on PLE. There, it meant a final deliverable, a complete system. Here it refers to a component of a complete system.

Product line governance

Because of the size, importance, and visibility of the product line, governance of the product line has emerge as a major issue. Stakeholders include multiple agencies of the US Government, inside and outside of the Department of Defense. They, and Lockheed Martin, had to create a number of governance structures, artifacts, and policies, to coordinate and prioritize the various and (at times) competing interests in each of the configurations, as well to head off confusion around who in the government was giving direction to Lockheed Martin, by what authority, and what direction was being provided [6].

Some of the more germane facets of governance include:

- **Regular, predictable build rhythm.** There are three build releases a year, one in each of January, May, and September. This so-called "1-5-9" rhythm brings great stability to the program. Everyone, inside Lockheed Martin or out, can expect and plan for the next release. Inside Lockheed Martin, build meetings are held weekly, to make sure the next release is on track. Not every customer configuration in the product line is required to accept every release; each makes its own decision according to operational needs and the content of the particular build.
- **Requirements review cycle.** As in all product lines, changes to requirements (whether to add new capabilities or address defects) that are made for one configuration may have intended and unintended impacts to other configurations, and must be reviewed across the product line with that in mind. A rigorous Requirements Review Cycle for programs is held in March, July, and November (a "3-7-11" rhythm) and is a joint Lockheed Martin/customer exercise.
- **Governance boards.** While Lockheed Martin worked to establish processes to manage a coordinated planning approach and day-to-day development activities to serve multiple masters, the government put in place a structure and associated processes to ensure clear and consistent direction is being provided to maximize the probability of success for all programs. This structure provides support for decision making at three levels: Strategic, Programmatic, and Technical. These boards adjudicate cross-program priorities and activities. Their respective decisions are captured in a Development Fielding Plan, a Branch and Merge Plan, and a Build Plan.

If a program introduces an upgrade or new capability, it pays for it. Other programs are free to pick it up, or not, as they wish, but they pay for any testing that is required and unique to their context. After a development is complete and time has elapsed, newly found defects become difficult to associate with any one program. In these cases all programs pitch in to pay to have the defect corrected. Lockheed Martin gets a special funding account to fix all defects, across the entire product line, that are not related to unique capability content in development. Any program receiving special development funding pays for defects in that development, up to its demonstration milestone, at which point the cost-sharing approach kicks in.

This, then, is the institutionalized PLE situation at Lockheed Martin when Agile joined the picture. As we shall see, some of these governance structures have been key to a seamless Agile integration.

Agile Development

Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change [16].

Although are many specific agile development methods, all rely on a small number of fundamental concepts:

- **Iterative, incremental and evolutionary.** Most agile development methods break the tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders.
- **Efficient and face-to-face communication.** Each agile team should include a customer representative (called the "product owner" in the scrum methodology). This person is appointed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer mid-iteration questions.
- **Very short feedback loop and adaptation cycle.** A common characteristic in agile is the daily "stand-up", also known as the daily scrum. In a brief session, team members report to each other what they did the previous day toward their team's sprint goal, what they intend to do today toward their team's sprint goal, and any roadblocks or impediments they can see to their team's sprint goal.
- **Quality focus.** Specific tools and techniques, such as continuous integration, automated unit testing, pair programming, test-driven development and other techniques are often used to improve quality and enhance project agility.

Scrum

Scrum is the specific agile approach chosen by Lockheed Martin. It provides a structure of roles, meetings, rules, and artifacts. Teams are responsible for creating and adapting their processes within this framework. Scrum uses fixed-length iterations, called sprints, which are typically 1-2 weeks long.

A key principle of scrum is its recognition that during production processes, the customers can change their minds about what they want, and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner [17].

There are three core roles defined in the scrum framework [17]:

- **Product owner.** The product owner represents the stakeholders and is the voice of the customer. The product owner writes (or has the team write) customer-centric items (typically user stories), ranks and prioritizes them, and adds them to the product backlog. The product owner should be on the business side of the project, and should never interfere or interact with team members on the technical aspects of the development task.
- **Development team.** The development team is responsible for delivering potentially shippable increments of product at the end of each sprint. A team is made up of 3–9 individuals who do the actual work (analyze, design, develop, test, technical communication, document, etc.). Development teams are cross-functional, with all of the skills as a team necessary to create a product increment. The development team in scrum is self-organizing, even though there may be some level of interface with project management offices.

- **Scrum master.** Scrum is facilitated by a scrum master, who is accountable for removing impediments to the ability of the team to deliver the product goals and deliverables. The scrum master is not a traditional team lead or project manager, but acts as a buffer between the team and any distracting influences. The scrum master helps ensure the team follows the agreed scrum processes, often facilitates key sessions, and encourages the team to improve.

A sprint is the basic unit of development in scrum. The sprint is a time boxed effort; that is, it is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.

Each sprint starts with a sprint planning event that aims to define a sprint backlog, identify the work for the sprint, and make an estimated commitment for the sprint goal. Daily scrum meetings are held to assess progress and identify impediments to reaching the sprint's goals. Each sprint ends with a sprint review and sprint retrospective, that review progress to show to stakeholders and identify lessons and improvements for the next sprints. [17]

Agile Comes to Lockheed Martin

Lockheed Martin has been able to achieve hundreds of millions of dollars in cost avoidance from their successful application of PLE [7]. But in late 2014, corporate management decided that Agile was the way to strengthen Lockheed Martin's competitive position through quality and affordability. This goal was levied on the entire organization, including, for example, Lockheed Martin's Orion spacecraft project, which involves some 350 engineers. Training began throughout the corporation in March, 2015.

Improvement measures for Agile are somewhat hard to come by, but the Standish Group claims that Agile projects are three times as successful as Waterfall projects (Figure 3). Others say "the use of agile methods results in increased cost-effectiveness, productivity, quality, cycle-time reduction, and customer satisfaction ranging from 10% to 100% [11].

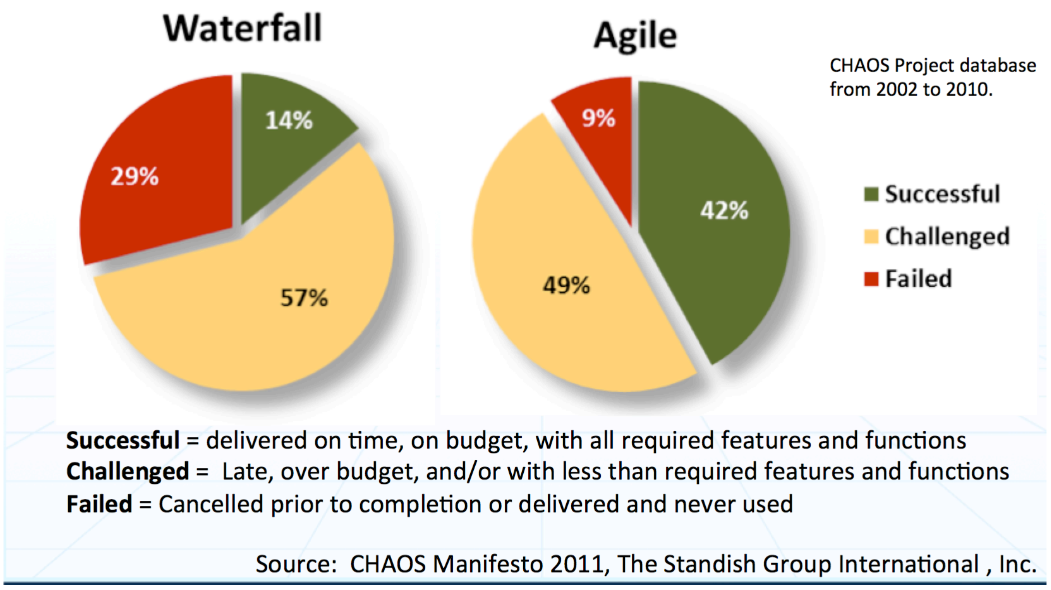


Figure 3: Agile projects are three times more successful than Waterfall projects according to the Standish Group.

These and other ROI results motivated the adoption of Agile. It fell to those parts of the organization already addressing competitiveness through affordability and quality with PLE to work out how to add Agile to the mix, and achieve still more.

Melding Agile and PLE

Agile is, by and large, about

- a (usually small) single product
- fielded by a single (usually small) organization
- by one or more (always small) teams
- in a series of extremely short iterative development cycles

In the product line being described:

- The products – plural – are large by any standard, comprising some 10 million lines of code and costing tens to hundreds of millions of dollars.
- The teams, spread across the products, are sizable. The largest component, for example, involves over 200 engineers. Some 800 engineers are involved in the product line overall[2].
- The build cycles, under the product line's 1-5-9 build tempo, are four months long.

How can these very different approaches be integrated?

Lockheed Martin's first answer was "incrementally." Figure 4 outlines a three-year phased approach to bring in Agile practices. The first phase concentrates on risk reduction, training, and release planning; the first release planning event was completed only a few weeks before this paper was submitted. The second phase continues to roll out and begins to to optimize workflows and team organizations that were put into place in the first phase; it also begins to re-define some of the customer interactions that need to change, such as adapting to finer-grained scheduling and budgeting and providing more embedded feedback. Finally, the third phase continues the roll-out, incorporates business coordination, and transitions from introduction to optimization.

But the question remains: How to bridge the gap between large-scale PLE and small-scale short-iteration Agile? Here there are three answers, each detailed in a sub-section below.

Small projects

Agile in general, and Scrum in particular, are about the lean execution of development *projects*. In the subject product line, projects are associated with component *products*. That is, the projects are already defined. The goal, then, became making those projects Agile. That in turn involved making the project teams into small, Agile teams working in short iterations to support small-scale tasks that, taken as a whole, sum up to the purpose and work of the original product teams.

[2] This product line turns out to be the largest application of Agile in the entire corporation.

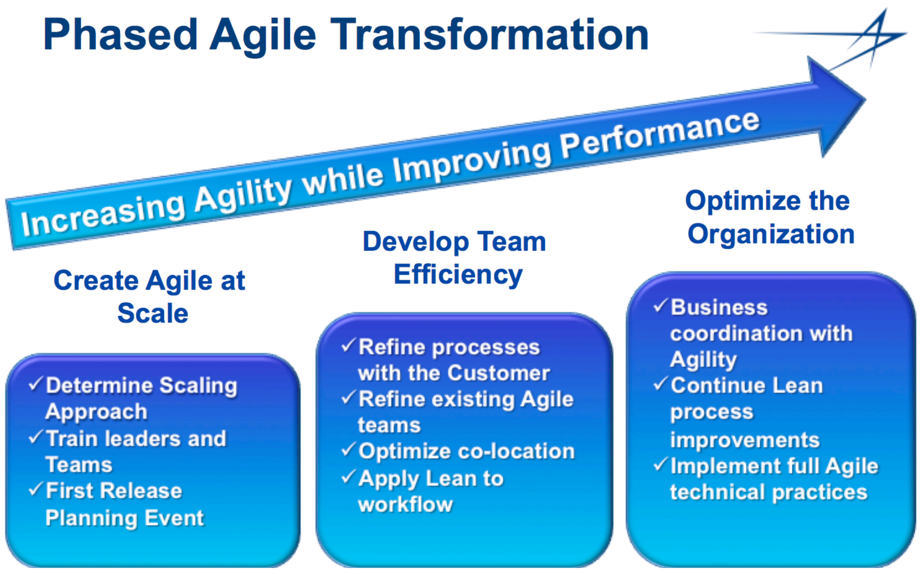
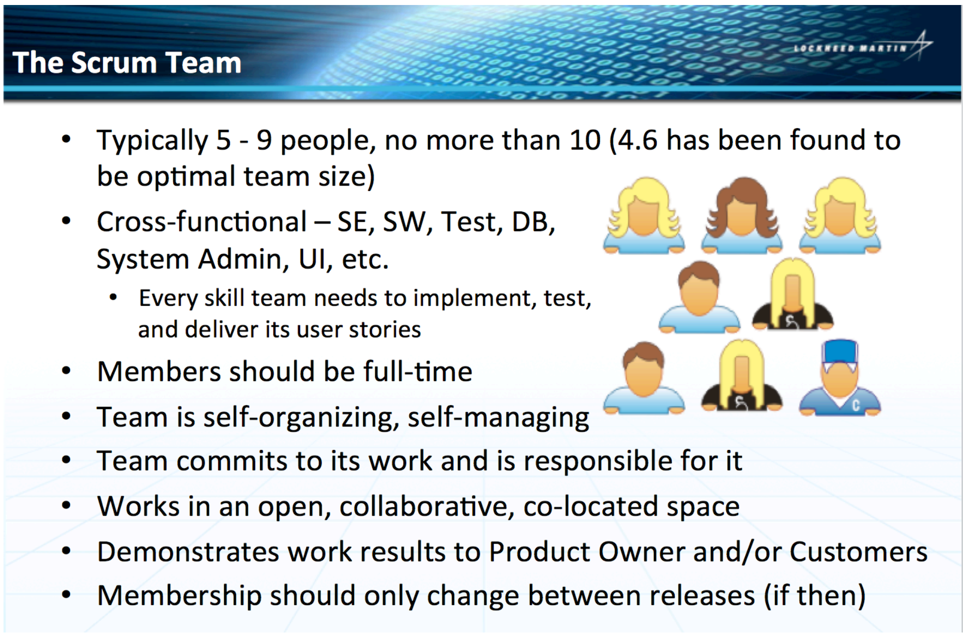

Phased Agile Transformation
Increasing Agility while Improving Performance
Create Agile at Scale
✓Determine Scaling Approach
✓Train leaders and Teams
✓First Release Planning Event
Develop Team Efficiency
✓Refine processes with the Customer
✓Refine existing Agile teams
✓Optimize co-location
✓Apply Lean to workflow
Optimize the Organization
✓Business coordination with Agility
✓Continue Lean process improvements
✓Implement full Agile technical practices

Figure 4: Lockheed Martin's three-year three-phase approach to incorporate Agile methods

Small teams

Small teams resulted from decomposing the products' large teams into Scrum teams of size seven to ten. The teams, considered together, do the same work as they did before, but the work has also been divided into smaller bite-sized chunks that match the team size and short iteration (sprint) schedule.

The teams are self-organized by product, mostly around areas of domain expertise or functionality, or around specific elements in the architecture. Figure 5 describes the essentials of individual teams.



The Scrum Team

- Typically 5 - 9 people, no more than 10 (4.6 has been found to be optimal team size)
- Cross-functional – SE, SW, Test, DB, System Admin, UI, etc.
  - Every skill team needs to implement, test, and deliver its user stories
- Members should be full-time
- Team is self-organizing, self-managing
- Team commits to its work and is responsible for it
- Works in an open, collaborative, co-located space
- Demonstrates work results to Product Owner and/or Customers
- Membership should only change between releases (if then)

Figure 5: Scrum teams

So far so good: Decomposing large teams into small ones, and large tasks into small tasks, is relatively straightforward.

However, instead of nine large product teams, the decomposition resulted in 100 or so small ones. Further, the twelve major configurations are, at any point in time, each in a different phase of development. For some, the major activity for the next release is writing concept papers; for others it might be writing code; for still others, it could be testing. Coordinating these activities among nine large teams was challenge enough; with 100 small teams, ensuring everyone is working towards common large-scale goals, without devoting all their time to management overhead tasks, becomes a concern of the first order.

This issue is Ground Zero of where PLE meets large-scale multi-project Agile.

To address this problem, Lockheed Martin has turned to the Scaled Agile Framework (SAFe), a “knowledge base for implementing agile practices at enterprise scale.” It defines the “individual roles, teams, activities, and artifacts necessary to scale agile from the team, to teams of teams, to the enterprise level.” [12]

Dependencies are tracked using Atlassian’s Jira software project and issue tracking tool. Jira helps plan and manage sprints, and it allows every team member to be involved in the planning. This involvement lets everyone see and be aware of task dependencies, and leads to more accountability, ownership, and buy-in to the team’s tasking. Teams vote collaboratively to agree on the scope of a piece of work. If priorities are not being met, then the higher-level governance structures describe earlier for the product line as a whole come into play to adjudicate conflicts and re-orient the work.

This agile management of teams also provides flexibility in work assignments. A team with more resources and less schedule pressure at the moment can “farm out” their services to other teams in need.

This approach, using SAFe, received its first serious trial this past September with the product line’s first full-up release planning event. All 100 teams, representing all 800 engineers, participated in this 3-day planning exercise and produced a coordinated plan for the next eight weeks. Jira provided the way to manage links among program and product work content, and all 800 engineers (many geographically dispsered) could see this picture unfolding in real time.

Small iterations

While inter-team coordination is squarely in the realm of SAFe, work inside each team follows classic Scrum, meaning that planned work is organized into two-week sprints. A certain number of sprints add up to a release.

How many sprints? Recall from our overview of product-line-level governance that Lockheed Martin has adopted a four-month release cycle referred to as their “1-5-9” rhythm. Thus, eight sprints constitute a release. Here is an elegant case where PLE and Agile work in synchronization with each other.

The work accomplished in each sprint is, again from Scrum, defined in terms of *user stories*. A user story is a “developer-sized” task that can be implemented and tested in one or two days. It constitutes the smallest capability that has business value.

User stories may be decomposed into *tasks*, which generally take 2-8 (worst case, 16) hours to complete.

Users stories are used to accomplish *features*. A feature is a sort of large-scale user story that often applies to an entire release. A feature is accomplished within a release cycle, and often spans teams in its scope.

Features in turn constitute *epics*, a capability that spans multiple releases (perhaps even over multiple years) and often involves the addition of major capability to the product line.

Figure 6 summarizes.

Operationally, each product team takes a program feature and decomposes it into features that affect their product. These product features are then decomposed into user stories and tasks. These constitute the team's backlog. The work under this approach is by and large what it would have been under the "pure" PLE approach, except that now it is decomposed into assignments that can be agilely managed with greater precision and predictability, and allowing for a much finer-grained prioritization.

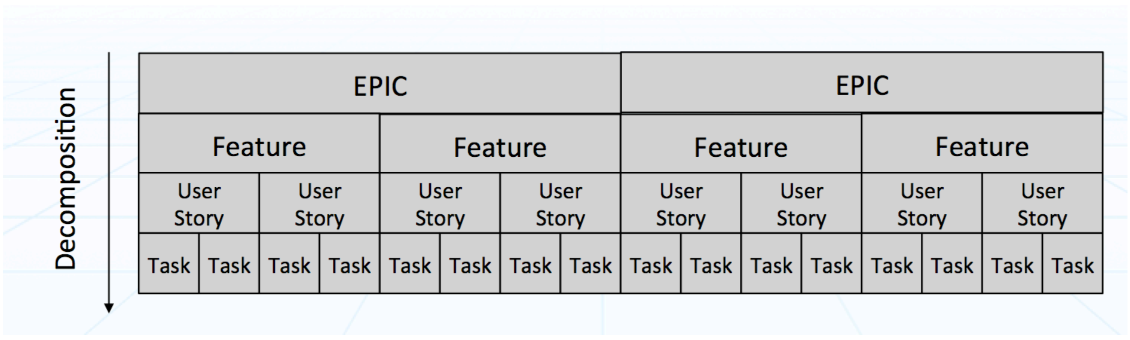


Figure 6: Epics, features, user stories, and tasks

Following Scrum prescription, each sprint includes daily scrum meetings and culminates in a sprint demo/review and a sprint retrospective. Table 1 outlines the contents and guidelines for each.

Table 1: Sprint events

Daily scrum meetings	Sprint demo/review	Sprint retrospective
• Display Visible Progress Indicators for team to review • Scrum Guidelines • Daily stand-up • 15 minutes • Not for problem solving • Three questions: What did you do yesterday? What will you do today? Is there anything in your way? • Schedule follow-on meetings to discuss details. • Whole world is invited • Only team members, Scrum Master and Product Owner can talk • Scrum Master writes down and tracks issues and requests for additional conversations	• Team "shows off" working product resulting from current Sprint (what is "done") • Typically takes the form of a demonstration of new features or underlying architecture; establishes trust with customer and stakeholders • Informal: 2-hour prep time rule; no slides, unless central to showing work • Whole team participates • Invite the stakeholders • Demo System • Invite BD	• Short meeting held at the end of each Sprint • Celebrate the completion of the Sprint • Take a look at what is and is not working • Team reflects on how to become more effective • Then tunes and adjusts its behavior accordingly • Whole team participates • Often uses a Start/Stop/Continue approach • Whole team gathers and discusses what they'd like to… Start, Stop and Continue • After the first few iterations, start using measurements

Conclusions

So far, Lockheed Martin is finding that adding Agile to the PLE mix is paying off. For instance, an agile approach to requirements allows them to adopt a "just enough, just in time" approach with just enough detail to size a project and ensure its technical and economic feasibility. This is a vast improvement from the "specify everything up front" approach that sometimes led to poor decision making and inadequate early insight into the requirements' downstream ramifications. Developing iteratively also allows users early access to the system to uncover new and evolving requirements.

Agile provides for more level loading and resource allocation. Previously, the response to a looming deadline was to "surge," adding resources for a milestone and then ramping back down. Now, with better planning and tighter customer involvement, that can avoided.

Lessons about teaming are emerging. First, teams should be co-located, if possible. Second, this structure can expose weaker individuals; everyone needs to carry their weight, since there's no place for mediocre performers to hide on a small team. Not everyone is cut out for this approach, as it requires individuals to perform to their best abilities.

Culminating each sprint with a review or demo for the customer (typically showing off new features or architectural improvements) establishes trust and instills confidence in the customer and other stakeholders.

Do Agile and PLE go together? The verdict is overwhelming positive at this point. Overall, Lockheed Martin views Agile as a way to optimize the performance of PLE teams. Agile complements PLE. Whereas PLE defined the teams, Agile takes that as a starting point and refines those teams and optimizes the performance of their members. Agile, in a sense, is an overlay atop the team and governance structures put in place by PLE. It provides a structure and coordination tooling for the work that these teams needed to do anyway.

Future work includes collecting and analyzing metrics to spot areas of concern and further improve performance. Classic agile metrics include *velocity* (essentially a measure of work accomplished against user stories in a sprint), which can lend itself to trend analysis over time. Also, the product line has made extensive cost avoidance measurements, and those will be monitored carefully as Agile achieves enough of a track record to observe how it affects those measurements. All signs point towards a significant and positive contribution.

References

[1] Brownsword, L. & Clements, P. A Case Study in Successful Product Line Development (CMU/SEI-96-TR-016, ADA315802). Pitts- burgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. http://www.sei.cmu.edu/publications/documents /96.reports/96.tr.016.html.

[2] Clements, P., Gregg, S., Krueger, C., Lanman, J., Rivera, J., Scharadin, R., Shepherd, J., and Winkler, A., "Second Generation Product Line Engineering Takes Hold in the DoD," Crosstalk, The Journal of Defense Software Engineering, USAF Software Technology Support Center, 2013, in publication.

[3] Clements, P.; Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002.

[4] Dillon, M., Rivera, J., Darbin, R., Clinger, B., "Maximizing U.S. Army Return on Investment Utilizing Software Product-Line Approach," Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), 2012.

[5] Flores, R., Krueger, C., Clements, P. “Mega-Scale Product Line Engineering at General Motors,” Proceedings of the 2012 Software Product Line Conference (SPLC), Salvador Brazil, August 2012.

[6] Gregg, S., Scharadin, R., LeGore, E., Clements, P. “Lessons from AEGIS: Organizational and Governance Aspects of a Major Product Line in a Multi-Program Environment,” Proceedings, Software Product Line Conference 2014, Florence, Italy, 2014.

[7] Gregg, S, Scharadin, R., Clements, P. “The More You Do, the More You Save: The Superlinear Cost Avoidance Effect of Systems and Software Product Line Engineering, Proceedings Software Product Line Conference 2015, Nashville, 2015.

[8] Kang, K.; Cohen, S.; Hess, J.; Novak, W.; & Peterson, A. “Feature-Oriented Domain Analysis (FODA) Feasibility Study” (CMU/SEI-90-TR-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[9] Krueger, C. and Clements, P. “Systems and Software Product Line Engineering,” *Encyclopedia of Software Engineering*, Philip A. LaPlante ed., Taylor and Francis, 2013, in publication.

[10] Linden, Frank J. van der, Schmid, Klaus, Rommes, Eelco. *Software Product Lines in Action*, Springer, 2007

[11] Rico, D. F., “What is the return of investment (ROI) of agile methods?” http://www.afei.org/WorkingGroups/ADAPT/Documents/rico08a[1].pdf, 2008.

[12] Scaled Agile Framework, www.scaledagileframework.com

[13] Software Engineering Institute, “Benefits and Costs of a Product Line,” http://www.sei.cmu.edu/productlines/frame_report/benefits.costs.htm

[14] Software Engineering Institute, “Catalog of Software Product Lines,” http://www.sei.cmu.edu/productlines/casestudies/catalog/index.cfm

[15] SPLC Product Line Hall of Fame, http://splc.net/fame.html

[16] Wikipedia, “Agile software development,” https://en.wikipedia.org/wiki/Agile_software_development

[17] Wikipedia, “Scrum (software development),” https://en.wikipedia.org/wiki/Scrum_(software_development)

Biography

- **Susan P. Gregg** is a Principal Project Engineer for the Lockheed Martin Corporation. She holds a B.A. in Physics from Rutgers University. She has over 30 years experience is systems and software engineering. She is currently the Technical Director for the US Navy’s Common Product Line.
- **Rick Scharadin** has over twenty years of Senior Program Management experience related to complex large scale system development, open architecture designs, software product line development, product integration, test and delivery for various Navy Aegis Baselines. He has accumulated over his career with Lockheed Martin 12 service awards, including manager of the year in 2001. Rick has a BS in Electrical Engineering from Penn State and a MS in System Engineering from Stevens Institute.

- **Dr. Paul Clements** is the Vice President of Customer Success at BigLever Software, Inc., where he works to spread the adoption of systems and software product line engineering. He was previously at Carnegie Mellon's Software Engineering Institute, where for 17 years he worked in software product line engineering and software architecture documentation and analysis. Clements is co-author of three practitioner-oriented books about software architecture as well as the field's leading text on software product line engineering.