A Methodical Approach to Product Line Adoption

Michael Dillon
US Army PEO STRI
12350 Research Parkway
Orlando, Florida USA
+1 407 384 5307

Mike.Dillon@us.army.mil

Jorge Rivera
Rowland Darbin
General Dynamics C4 Systems
12001 Research Parkway, Suite 500
Orlando, FL 32826
+1 407 275 4820
Jorge.Rivera@gdc4s.com
Rowland.Darbin@gdc4s.com

ABSTRACT

The evolution of the U.S. Army's Live Training Transformation (LT2) product line of combat training systems, including the move by the Army to consolidate management of the product line under a single contracting team, has provided a natural experiment that validates the hypothesis that product line engineering practices are more effective than traditional software engineering practices, and has demonstrated which product line adoption approaches are more successful than others. By analyzing this natural experiment, the product line team has been able to apply a methodical approach to product line adoption across the development organization and successfully adopt second generation product line processes. This paper explores that methodical approach. It will enumerate the steps that led to successes and explore the contributing factors and unintended consequences of failures along the way. Additionally this paper will explore how this approach is being employed to extend the LT2 product line beyond software.

Categories and Subject Descriptors

D.2.2 [**Design tools and techniques**]: *product line engineering, software product lines, feature modeling*

General Terms

Management, Design, Economics.

Keywords

Product line engineering, software product lines, feature modeling, feature profiles, bill-of-features, feature constraints hierarchical product lines, variation points, product baselines, product portfolio, product configurator, second generation product line engineering, product line governance, product line adoption

1. Introduction

Live Training Transformation (LT2) [12] is the product line strategy put in place by the United States Army Program Executive Office for Simulation, Training and Instrumentation (PEO STRI). Through the use of LT2, the Army's office of the Project Manager (for) Training Devices (PM TRADE) builds and maintains live training systems in support of homestation training, deployed training, urban operations training, Maneuver Combat Training Center training and instrumented live-fire range training. PM TRADE is the Army's acquisition agency for live training systems.

LT2 has realized significant improvements in cost savings and cost avoidance totaling hundreds of millions of dollars [3] in development and sustainment of live training systems. Live training systems in the product line support year-round training exercises at over 150 ranges across the globe, training individual soldiers as well as full brigades in live force-on-force and force-on-target engagements (Figure 1, Figure 2).

Figure 1 Soldiers experience a simulated improvised explosive device (IED) attack and perform live-fire qualifications, respectively, using training systems in the LT2 product line. (US Army photos)

SPLC 2014, September, 2014, Florence, Italy.
Copyright 2014 ACM 978-1-4503-2740-4/14/09…$15.00.
http://dx.doi.org/10.1145/2648511.2648550

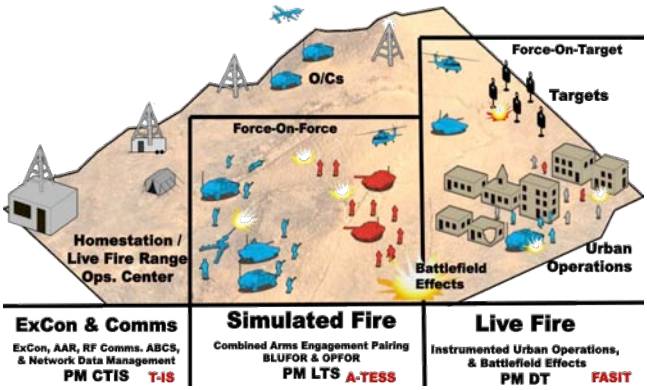

Figure 2 Live Training Domain. "ExCon" refers to exercise control (the part of the system that oversees and controls the training scenario) and "Comms" stands for communications.

Prior to the implementation of the LT2 product line, live training systems and devices consisted largely of products developed separately by a variety of different manufacturers to comply with disparate requirement sets and were designed and implemented without a common framework. Commonality was not attempted and interoperability among systems was rare, difficult, and costly to achieve. Configuration changes to both hardware and software were often performed on-site as part of the sustainment effort, making configuration control virtually impossible.

Over the past 14 years, the LT2 product line has evolved from a loosely associated group of contracts fielding U.S. Army live training systems, to a fully functioning and award winning[1] software product line.

In 2009 the Army issued a contract to consolidate the management of the LT2 Product Line, to gain further optimization in deploying live training systems. General Dynamics is the prime contractor, in partnership with experts in the fields of product line engineering and live training, to develop a methodical approach to second generation product line engineering [3] adoption. The contract is called Consolidated Product-Line Management (CPM). The successful adoption of second generation product line management is currently yielding $18M per year in cost avoidance for products generated from the shared baseline.

The evolution of LT2 as a product line is documented in numerous papers (for example, [1][3][5][6][7][8][9]) and is measured continuously though yearly contractor performance reviews. This unique situation of continuous critique and re-evaluation has provided the CPM team with a rich data set revealing a natural experiment in product line adoption. The path taken by LT2 has sometimes been fortuitous, sometimes planned, and sometimes less than ideal. As is the nature with all engineering efforts, there is little appetite for failure and the community of contractors supporting LT2 has amassed a great deal of domain expertise that makes genuine failure extremely rare.

To formalize the adoption of product line methodologies we have analyzed the past contracts and cast the LT2 evolution as a set of *execution methodologies*. Each of these methodologies is an approach to dealing with the disciplined management of commonality and variation, which is the essence of product line engineering.

The methodologies together paint a picture of stepwise PLE evolution from loosely coupled product development efforts to a streamlined, shared-asset-centric, automation-powered product line factory. Each methodology has its advantages and disadvantages, which will be identified and discussed.

These methodologies are not mutually exclusive; it is not the case that one fully ends before the next begins. Because Live Training efforts and the LT2 product line are continually expanding, each methodology may well continue to play a role even after the primary focus has shifted to another. By characterizing these methodologies using concrete examples that are clearly articulated and familiar to the product teams, we are able to build awareness and formalize the process of improvement, subsequently evolving to a greater level of efficiency in product line engineering.

In the course of this paper, we will evaluate several product line failures. These did not result in failures in product delivery but have revealed opportunities for improvements in product line engineering.

For each methodology we will identify attributes that enable the methodology, define the goodness that execution within this methodology provides as well as the negative side effects. We will expound on how this methodology exhibits itself within the LT2 ecosystem and what successes and failures have resulted from executing under this methodology in the past.

The CPM team defined five execution methodologies:

- Isolation
- Awareness
- Shared Delivery
- Shared Baseline
- Collaborative Development

In the LT2 product line, the current desired execution methodology is predominantly collaborative development. This has been achieved to varying degrees across most efforts within the LT2 product line.

Four representative LT2 product line products provide specific context for this paper, they where chosen for their breadth of capabilities and scale of usage and deployment. These range from individual soldier training to Brigade Combat teams of thousands of soldiers, and live fire single range paper targets to complex automated system of instrumented targets, as well as simulated engagements between instrumented forces. The representative systems are:

- Combat Training Center Instrumentation System (CTC-IS) provides advanced collective force-on-force and live fire training to Brigade Combat Teams and Echelons above Division in realistic battlefield conditions.
- Digital Ranges Training System (DRTS) supports live fire exercises and individual and crew-served weapon skill qualification or sustainment, and collective Training Events.
- Homestation Instrumentation Training System (HITS) provides Battalion/Taskforce and below, live combined arms Force-On-Force training exercise and test events at home stations and deployed training sites.

[1] Recent awards include: 2012 U.S. Army Acquisition Excellence Award for Information Enabled Army, 2012 U.S. Army Modeling and Simulation Office (AMSO) Award, and 2011 National Training and Simulation Association (NTSA) Award.

- Targetry Range Automated Control and Recording (TRACR) System is a software product that supports the planning, execution, and review of scenario based training at non-instrumented army training ranges.

2. Isolation

The most natural execution methodology is for product teams to operate independently. The state is where many, if not most, product line organizations find themselves at the beginning. However, our experience is that this may continue to occur, in pockets, within organizations regardless of the presence of a product line or product teams. Further, this model can be an effective one when used appropriately [10]. Isolated efforts typically focus on unique technology that does not already exist in the product line.

The U.S. Army often fosters isolation by issuing independent contracts for systems even if there is potential requirement overlap with adjacent projects. These contract boundaries were in place prior to the advent of LT2, and remain in place even with the highly effective LT2 product line. Isolation is also common in efforts that are outside of the purview of the organization management. In the commercial world, this could come about because of a merger or acquisition; in the government-contracting world this is typically due to a contract from a different but "nearby" program office asking for work in or close to the domain of the product line. As can be seen from the 4 representative products selected for comparison in this paper, it's simple to draw logical boundaries around systems based on size, deployment style or training objective. Although LT2 has been successful in consolidating these products, often the organizations responsible for initial contracting don't recognize the potential capabilities overlap. Because these contracts are not mandated to participate in the product line they are sometimes implemented independently.

2.1 The Good

When executing independently, teams have the freedom to operate without the constraints imposed by other teams. Teams can effectively control their own destiny. During execution decisions are made quickly to meet the needs of specific and familiar customers and end users. Opportunistic reuse occurs through dependency selection and inheritance from previous efforts. Teams develop expertise that improves their efficiency – albeit efficiency at producing an isolated product – during the development cycle. Rubin et al. cite low upfront investment, reuse of verified code (and other types of engineering assets as well), rapid development, and independent development as advantages of this approach [10].

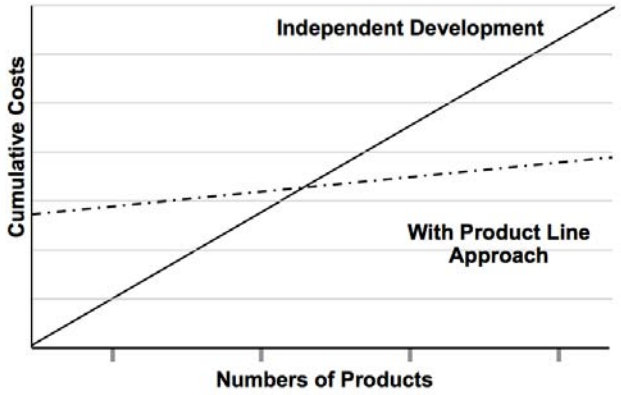

Figure 3 The cumulative cost of independent development vs. product line development

2.2 The Bad

The product line literature has been vocal and strident about the disadvantages of independent development; in fact, product line engineering's reason for being can be seen as overcoming the deficiencies of development based on one-time reuse [4][13]. Fundamentally, the problem is that reuse happens when a project starts, but the commonality among the various products is not exploited after that. The products, instead of continuing to share assets and gain advantage from the commonality, spiral off on their own evolutionary trajectories. The organizational effort for a group of *n* similar project is *n* times the cost of a single project. Adding and staffing a new project costs the same as staffing the first project, whereas under the product line paradigm the cost of a new project adds only a marginal cost. This understanding is represented in the familiar graph of product line economics, which first appeared in [14], shown in Figure 3.

A drawback of this approach, often overlooked, is that the price of isolation may never be recognized. Product teams are accountable only to themselves and their customers. If the customer is happy, then there is no incentive to change and if the customers remain, the product thrives. However, advancement of the product is dependent entirely on the single product team. If the team, or key resources, are pulled away to support another project, or if funding is no longer available, advancement of the product stagnates. Other competitive teams may dominate the market space, even competition from within the organization.

From an individual product execution standpoint, isolation is often not viewed as failure. If the product line engineering approach is unknown, why would isolated teams do anything else? But from a larger organizational standpoint there is failure. The failure is not in the successful fielding of a system – the systems are successfully fielded – or the effectiveness of the training provide by the system – the training is extremely effective. The failure is the missed opportunity for future sharing and exploitation of commonality, and in the additional cost required to re-implement existing capabilities. Prior to and even at the beginning of the LT2 product line, this approach was acceptable, but now the occurrence of these efforts is, we find, typically due to lack of education on the contracting side or the not-invented-here pride of the executing team.

2.3 Use in LT2

LT2 received a requirement for a sound effects simulator project, which was centered on adding a new capability to generate realistic battlefield sounds at very high decibel levels. LT2 responded by encapsulating that new capability, establishing an expert team for quick execution, and leveraging standards to ensure that the final product could be incorporated without negative impact into the (one) existing product that needed it. Even under the umbrella of a very successful product line effort, this capability was developed in a product-specific way to serve the needs of that product in the timeliest way possible.

3. Awareness

The first step in migrating to a product line execution strategy is to ensure there is awareness of other teams operating within the domain. Awareness does not mean launching an internet web search to discover what other teams are doing and trying to leverage their experiences. Awareness begins when organizations communicate and share information among teams on a regular, fostered, and repeated basis. Within organizations, information sharing typically starts with teams sharing non-code artifacts such as study results, or loaning experts from one team to another.

Awareness includes action, and is achieved when influences between teams ultimately affect the implementation of delivered products.

3.1 The Good

Effective awareness results when products are aligned with each other in both implantation and philosophy. Awareness encourages adoption and compatibility though standardization of interfaces, agreement on data types and design strategies. Awareness doesn't eliminate, but does reduce, the re-implementation of high profile system capabilities such as critical algorithms, presentation layers and external system integration. Sharing of systems' libraries and dependencies are encouraged. Because teams are operating independently, the impact to change is self-contained. Decisions about when to incorporate external inputs are weighed against internally defined performance goals.

Awareness need not happen under a heavy management hand. Clements and Northrop recount the case of product line sharing that began when two friends and colleagues, managing separate but similar projects in the same company, casually discovered the repetitive work they were each doing while having lunch together. They decided to cooperate and collaborate rather than continue to work independently, and a product line was born ([2], sidebar "Lunching and Institutionalizing,").

The discovery meetings held under the auspices of an awareness regime can also be seen as a dress rehearsal for the kind of regular coordination meetings that are required under a full-fledged product line effort.

3.2 The Bad

The initial benefits of awareness can mask the downsides. Teams still can incur high amounts of rework. Reuse is touted but is opportunistic. Typically, independent teams encounter and solve the same problems without knowing it. Because teams are looking forward to solve the next problem that they encounter, under an informal awareness regime little feedback is provided to other teams who have yet to solve the same problem. The overhead of meetings to keep all teams up to date can become burdensome, especially as deadlines approach and already taxed staff focus on their immediate assignments. Such meetings can also yield little benefit since the reuse they might foster is, again, opportunistic.

3.3 How to Enable

Despite its drawbacks, awareness is still a step up from independent development and should be nourished on the way to bigger and better things. The lunching example notwithstanding, the most effective movement of teams from isolation to awareness requires active intervention by higher level management. Though sometimes teams seek outside inputs and perspectives on their own, this is not the norm since the focus on executing tasks at hand outweighs the expected return for time spent collecting external inputs. Thus, the need for management intervention. Teams do however have a natural inclination to share when they have something unique to contribute to the community (or when they perceive that the community has something to contribute to them!).

A common architecture for teams is a powerful lever to motivate awareness of commonality. Under a common architecture, teams will not be surprised to learn that opportunities to exploit commonality are rife. If they are able to begin the sharing that is possible, they can then to focus on the unique capabilities that define their system. This payback motivates further sharing (which, under the awareness paradigm is still voluntary and opportunistic and needs to be nurtured since it is not yet planned or mandated). This helps to enforce the unique contributions of teams giving back to the community and provides a common language for communication.

In this way, the voluntary, community-aware awareness paradigm resembles the early "co-op" model of product lines from Hewlett Packard's Owen Firmware Cooperative [11].

3.4 Use in LT2

Within LT2 PEO-STRI established a dedicated contract to develop a Common Training Instrumentation Architecture (CTIA) that provided core life training capabilities enabling product teams to focus on the core mission of providing tailored training systems to end users. In Figure 4, the yellow bars indicate the quantity of each product's source code (x1M SLOC) provided by CTIA to accomplish common live training capabilities.

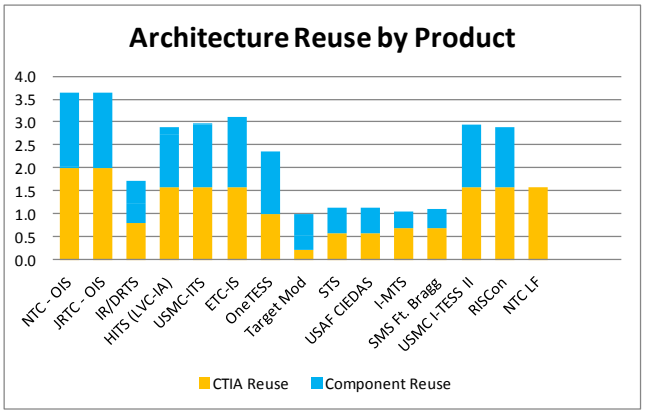

Figure 4 Reuse in LT2 Products under the CTIA common architecture

The LT2 Portal (Figure 5) is the primary mechanism for communicating information between live training product teams, including community news and events. Collaboration areas on the portal are tailored to specific topics and forums provide easy access to product teams. By providing a neutral communications path and quality information, the LT2 Portal has encouraged awareness and continues to greatly accelerate the development of training systems even outside of the LT2 product line.

Figure 5 The front page of the LT2 Portal (www.lt2portal.org)

Establishment of weekly Core Asset Working Group (CAWG) meetings has provided regular and frequent interaction between product teams. The value of the CAWG is in the formalization of

good engineering practices across the domain and the sense of community that frequently leads to relevant and productive communications between teams from different branches of government and disconnected, competitive contractors, that would otherwise have no mechanism for casual interaction at the executing level.

The LT2 Product Line also maintains a Standards Working Group that is dedicated specifically to interoperability standards, internal and external to the live training domain. This model of fostered awareness mimics other industry working groups. The downside to open community working groups are that they are typically attended by organizations either currently under contract to interface with the standard or external parties with a vested interest in influencing standard for future gain. Inactive contractors are sometimes marginalized with negative impacts. This is mitigated by timing events to coincide with contract proposal schedules and by dedicating technical engineers who participate in standards development as neutral parties.

4. Shared Delivery

Shared delivery occurs when all product teams provide their outputs to a common technology shelf for consumption by the community. A technology shelf could be a shared internal portal or an app store for a deployment platform. In the context of product line engineering, shared delivery almost always includes the source that generates the delivered products. When teams move from awareness to a shared delivery methodology they are intrinsically compared to each other. Differences between delivered products are presented such that they convey value, not as a product that meets a specific objective, but instead by how the components meet the goals of the domain.

4.1 The Good

Shared delivery means that products must meet a level of quality expected by the community. Teams begin to consume others' contributions to the technology shelf in an effort to reduce their internal development costs. Duplicate functionality is often eliminated by the product line teams picking the "Best of Breed" offering and encouraging the consolidation of that offering's capabilities into a single component. As interdependencies increase and multiple product teams incorporate common components into their products, more knowledge is shared across product teams. Reuse is more holistic in shared delivery, enabling substantial cost avoidance and return on investment when deploying new products from the shared collection.

The shared delivery paradigm also makes it much more easy for multiple contractors to cooperate, whereas simple awareness usually only applies within the walls of a single organization.

4.2 The Bad

Despite its advantages, shared delivery is still a paradigm based on opportunistic (instead of planned, strategic) reuse. Coarse-grained reuse in a shared delivery model is based on the clone-and-own of the previously released components in the technology shelf. Divergence of products occurs because teams fixing the same bugs and adding enhancements do not deliver their updates to the shared collection until their development cycle is complete. Depending on the magnitude of the changes, merging divergent components back together is often prohibitive. Back merging is acceptable when product teams execute sequentially or the number of teams is small. As the number of teams participating increases and the frequency of change is accelerated, or when development cycles for teams become very long in-between deliveries, this model quickly becomes unsustainable.

In the case of government contracting it's common for government owned source code to be provided as a contract deliverable; however, this source code is not easily distributed to other members of the community and is typically intended for handover to a subsequent contractor for post-deployment support and sustainment. This model makes it easy for contractors to remain the experts on the product they delivered, to the extent that it becomes very hard for other contractors to continue development or reuse the developed capabilities. By enforcing shared delivery as part of the statement of work when the contract is let, the contractor is held accountable for their delivered products and the community can obtain access without needing explicit information about the effort or the additional overhead of cost and schedule of getting delivered software handed over to another contractor.

4.3 How to Enable

Shared delivery is a natural and incremental extension to the awareness methodology, requiring only marginally more management mandate and community-directed volunteerism. Mandating product delivery to the common technology shelf must be a directed form the over-arching organization. The barrier to delivery must be very low to prevent the step of sharing being viewed as adding unnecessary burden. The organization must support the shared collection and keep it relevant to the community. Relevancy is enhanced by embedding the shared collection into an existing source of domain information such as a community portal, forum, or collaboration area. By making the delivered items an intrinsic part of the ecosystem, the shelf remains relevant and active.

4.4 Use in LT2

For the Live Training community the LT2 Portal is the technology shelf and has been a key contributor in breaking through the barrier to inter contractor sharing. By mandating delivery to the LT2 Portal in all statements of work the community has been able to significantly increase it's effectiveness in fielding training products. Because the community has access to established components that are targeted specifically to the domain, the customer has been able to issues contracts tailored to the development of specific training enhancements without having to burden themselves with the up front cost of deploying the basic capabilities that have been fielded numerous times before on other contracts. Figure 6 shows the number of reused common components that are fielded with each LT2 product. The largest systems consume the most common components, products tailored to smaller training environments are able to consume only targeted portions of the superset. LT2 has been able to field products composed entirely of reused code with little to no new development on the part of the fielding project. The recent contract to field an updated Live Fire system to the National Training Center leveraged capabilities, already deployed to small unit live fire ranges, to a battalion-level training site with minor updates. This targeted effort was able to leverage contractors that did not develop the initial capability and add the small portion of new requirements with very little investment and risk.

Failures for shared delivery are typically encountered when the technology shelf is too broad and the contributions do not adhere to a common standard.

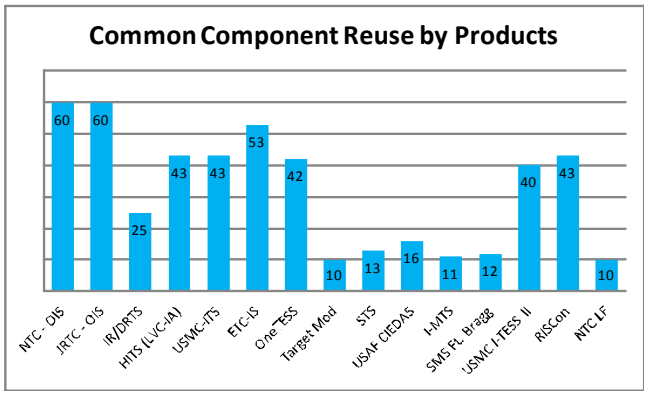

Figure 6 Component Reuse from Shared Delivery

For example, internal to General Dynamics an effort to reduce development and sustainment costs across projects ultimately resulted in less than expected adoption due to the extremely diverse nature of the contracts being executed. For example, blending of embedded satellite communications, battlefield simulation and air traffic control capabilities into training systems ultimately resulted in the technology shelf being unused by most and useful only for a small subset of contracts.

Other failures are typically due to product teams that refuse to contribute. These teams have a market that, although overlapping, is perceived to be uniquely better or different from the rest of the community. Examples like this in LT2 resulted in contractors fielding systems that are unsustainable by the community without significant additional investment. These systems are hindered due to the single project's funding to advance the system and subsequently get surpassed in capability and quality by the community supported product line.

5. Shared Baseline

As teams being to incorporate each other's products from the shared set of core assets under the shared delivery paradigm, the need to accelerate the inclusion of changes due to fixes and feature request drives to integrate earlier in the development cycle. Waiting for external teams to deliver products becomes a bottleneck.

This bottleneck is broken when baselines are merged into a single source repository[2] that all teams are concurrently working from to generate their products. The baseline is managed by a single organization that is responsible for baseline integrity. Consolidating configuration and data management for product teams into a single core team reduces the cost to individual products. Figure 7 shows the CTC and HITS development teams making changes to the share baseline of the 2D Map core asset, which is then incorporated into the release of the respective products.

Additionally, as product teams become more intertwined, requests for changes become request to make the change themselves and the lines distinguishing ownership of core assets becomes less clear.

[2] For LT2 the source repository is the one of most significance, but the point applies to repositories for all kinds of shared assets.

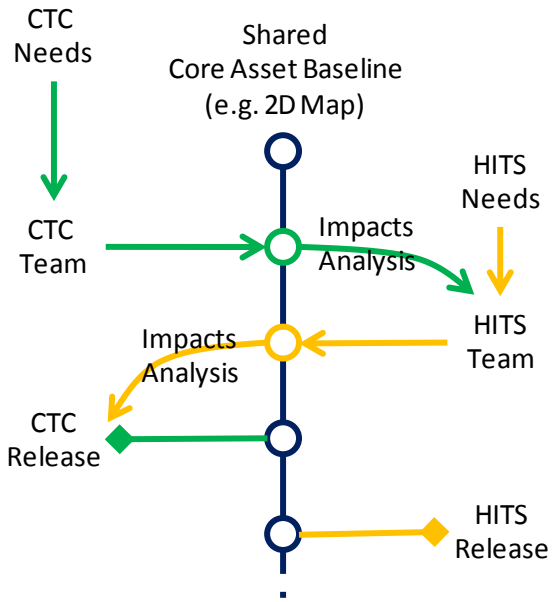

Figure 7 Changes to a Shared Baseline by multiple product teams

5.1 The Good

A distinguishing benefit of a shared repository is that product teams are no longer dependent on other teams' development cycles to release to a shard collection before getting the latest version of a common component. Fixes to the baseline are added as they are validated and become immediately incorporated into all consumers' products. As the number of consumers of the common baseline grow the amount of testing increases, detecting more defects, which in turn increases the quality of the baseline. Teams spend the time that was dedicated to developing duplicate capabilities to improving the baseline and adding enhancements. The baseline grows to support a larger community and the impact of re-composing capabilities into new products becomes much lower. Products targeted to small user communities, that would have been unable to independently fund a new product, now provide very high quality tailored training with minimal investment.

In addition, the blurring of the lines of ownership of the core assets is a precursor to true shared asset teams whose purview is the entire portfolio and not specific products. CPM has been executing the majority of the product sustainment activities from a shared baseline using features to encapsulate product specific variation. Currently this is enabling teams to expect 3 times the number of fixes returned to their baseline by other product teams than they would have executing independently. Defects found in the field have dramatically reduced and product quality is increased. This is shown in Figure 8.

5.2 The Bad

The role of the repository gatekeeper is critical to the integrity and responsiveness of the baseline. Releases from the baseline must be coordinated to ensure that all required changes are merged prior to the baseline freezes. Coordination of the changes and validation of changes becomes burdensome as products are frequently required to validate changes that were made by other teams who have not yet released. The overhead to meet stringent processes for committing to the baseline becomes excessively high, also developing teams frequently do not fully understand all possible

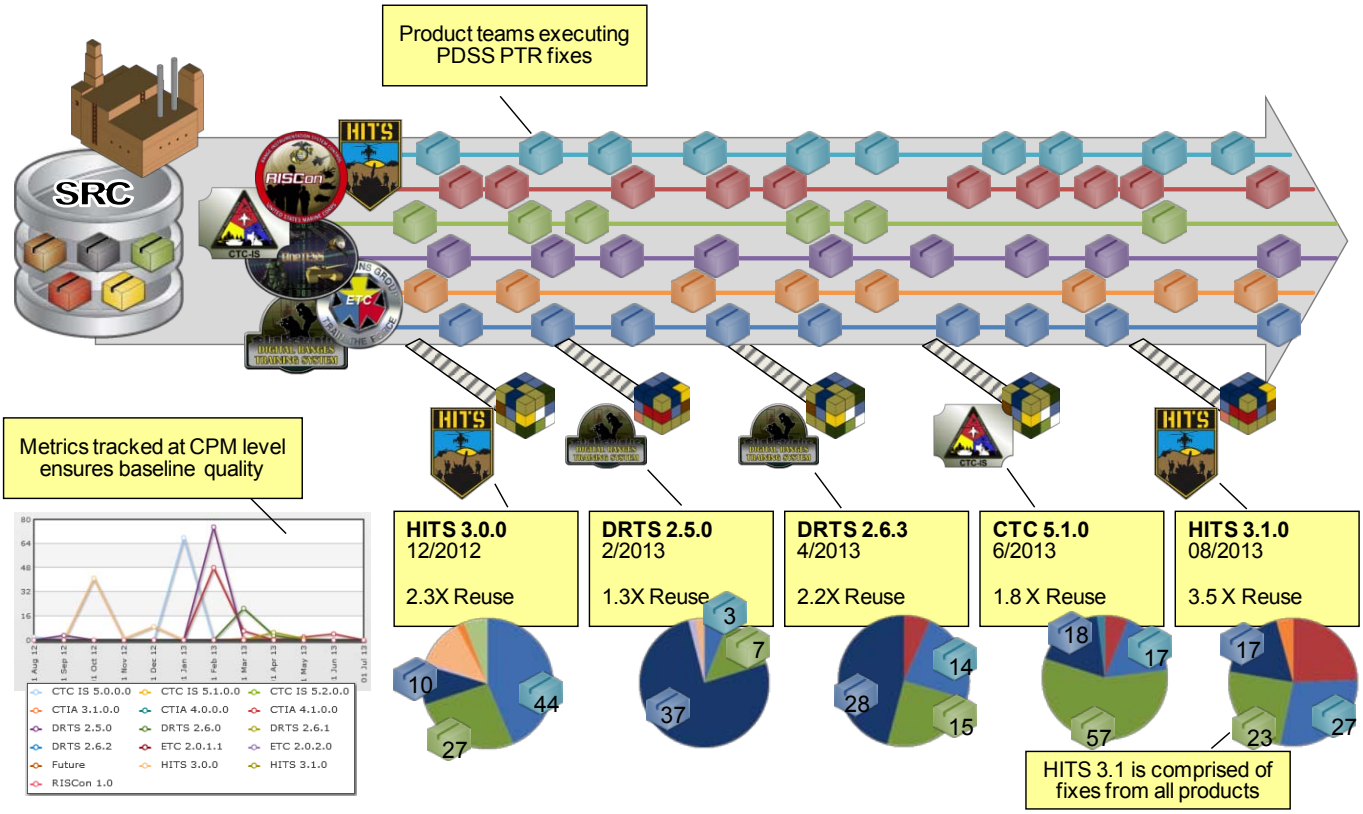

Figure 8 Product Releases from a shared baseline.

situations where their components will be incorporated into other products, resulting in undetected defects and additional rework. The number of coordination meetings is increased to mitigate these issues; this becomes a burden as the number of active product teams increases.

5.3 How to Enable

Migrating to a shared baseline requires having the process and tools in place to make it successful. CPM instantiated a community accessible central repository actuated from a common feature model. The shared assets are endowed with variation points, which are places in an asset where a product-specific instance must differ in order to support the specific features of a product. For LT2 we chose Gears as our feature modeling tool and product configuration [2]. Centrally managing product variation using feature models enabled product teams to continue executing without sacrificing the individually delivered products.

On top of the tools, LT2 established a strict set of processes including merging rules that prevents unintended changes making their way into the common baseline. Strict adherence by product teams and technical stopgaps to prevent unintentional baseline contamination are critical to product teams gaining the confidence necessary for the shared baseline to be successful. Having an open community that has already become indoctrinated through the Awareness and Shared Delivery methodologies makes gaining confidence less challenging; we believe that attempts to converge baselines prior to gaining this kind of confidence will be much more difficult and may not be successful.

In the case of LT2, we found pockets of shared baselines to already exist in the environment. Prior to CPM the "TRACR" and "HITS" product baselines were already shared across similar projects internal to the executing contractors. Merging the HITS baseline in with the Common CPM baseline meant establishing confidence between contractors where confidence in the smaller shared baseline already existed.

A significant concern when migrating to a common shared baseline is that the constituent members must perceive intrinsic value in migrating. Even with flawless technology and a high degree of baseline confidence there is risk associated with giving up control of your configuration management baseline to a third party. This risk must be lower than the benefit of the shared baseline. In LT2 this added value is clearly seen by the acceleration of fixes and features to the baseline as new teams join.

5.4 Use in LT2

Prior to the CPM program the baselines for the CTIA, CTC, and DRTS projects were maintained as separate baselines at the Integrated Development Environment facility. This is a government-run development facility providing an integrated lab for LT2 development and sustainment. Programs either have resident labs in the facility or utilize its capabilities remotely. As a badgeless facility that accommodates multiple contractors, the IDE provides improved collaboration, software P/L management, a co-located workforce, and significant cost reductions due to shared lab space, shared licenses, IT equipment, reduced facilities overhead, and resource sharing.

These baselines were loosely coupled. Both CTC and DRTS consumed and contributed to CTIA but the perceived value of complete consolidation did not outweigh the perceived risk. There was no single gatekeeper responsible for the baseline integrity and teams did not coordinate changes. Costly merges were the norm,

and component divergence proliferated. With CPM, the baselines merged and were controlled by a single configuration management team. The use of Gears enabled teams to encapsulate change and continue developing with minimal impact to their deployed configuration. Initially the value was low as only 3 teams were contributing changes, and changes were mostly isolated to each teams most important components. Due to earlier divergence, many of the most important components had capability overlap but were maintained independently.

At the advent of CPM, the DRTS, CTC, and CTIA baselines were fully merged into a common CPM Baseline. Additionally the HITS, ETC, and OneTESS baselines were merged into the CPM baseline and no longer independently maintained at contractor sites. The overlap of component reuse increased dramatically, as shown in Figure 9. Subsequently defect detection and fixes accelerated greatly increasing baseline quality and team confidence. The acceleration of positive change and increasing baseline quality tipped the scales of perceived value, greatly outweighing the perceived risk of the shared baseline. This trend continues with each additional team working from the shared baseline that now supports, on average, seven concurrent active products.

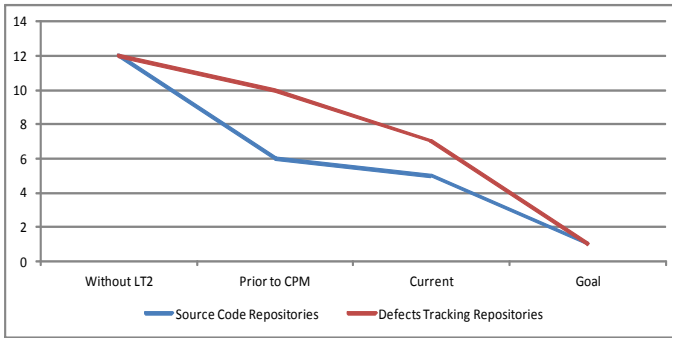

Figure 9 LT2 Repository Consolidation

6. Collaborative Development

When executing in a collaborative development methodology, teams are intertwined and function not as independent actors with individual product agendas but a single organization executing in concert to address the concerns of the community that the product line serves. Teams are centrally governed and directed form a consolidated tasking schedule. Implementations are strategic and evaluated with full consideration for planned reuse and multiple intended deployment configurations. Collaborative development means decisions about enhancements and fixes to implement are considered across the product line and evaluated against all available resources. Figure 10 shows the inputs from multiple product teams being triaged and prioritized by a collaborative development team. These needs are then provided to the implementation team for incorporation into the same shared baseline discussed earlier. Core Assets are released from the baseline to ensure that they support the needs of all consuming products without each product team needing to perform impact analysis on each baseline change.

Collaborative development represents the full realization of a product line engineering *factory* paradigm. Under this paradigm, shared asset engineering teams manage assets scoped for the entire product portfolio. As in the shared baseline methodology, feature models are used to exercise variation points in the assets to produce product-specific instances.

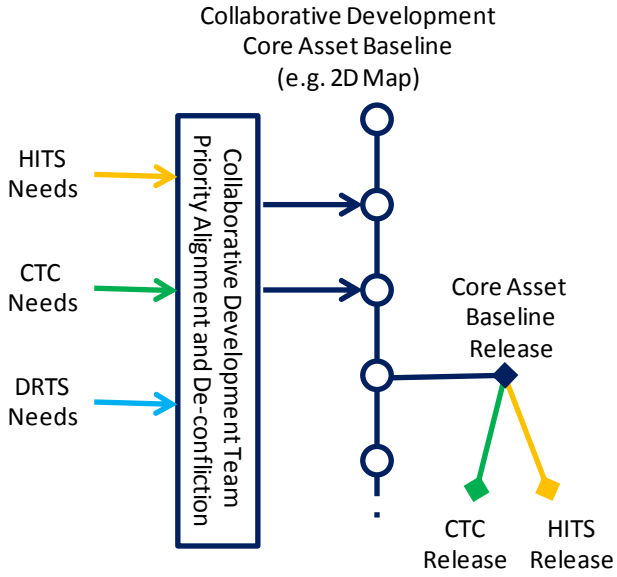

Figure 10 Collaborative Development supporting multiple products from a single baseline

6.1 The Good

By executing collaboratively, product teams focus on customer and end user needs. Needs are reported to the factory team, isolating products from the details of the implementation and enabling product teams to plan for longer term goals. The teams executing the changes are arranged so that they are focused on the expertise necessary to support the core assets in the product line rather than by historic product affiliation. By shifting accountability for deploying capabilities, factory teams ensure that the most qualified resources are applied to problems no matter what product team is funding the change. The efficiencies of this methodology are measured by affectivity across the entire product line and cost avoidance metrics are applied to encourage implementations that benefit the most products possible.

The factory paradigm is intuitive to understand and therefore easy to teach. It helps the entire organization have a common vision about how the organization does business and produces its portfolio. It also invites conversations and discussions about aspects of the product line outside the scope of "mere" development, such as how best to educate the customers about the product line approach and how cooperation among customers would benefit all customers.

Proactively cross-pollinating resources without product boundaries allows for members to be used more effectively while acknowledging that their expertise in a particular area is recognized and valued across the entire factory.

6.2 The Bad

The management of the baseline requires that all core asset teams remain in continuous communication and as the factory teams grow they can begin to encounter the same repeated implementations and boundaries as were evident in product centric development. To counter this tendency additional governance overhead must be expended by the factory to ensure that executing teams are coordinating.

6.3 How to Enable

Collaborative development means relinquishing the control of almost all product development to a factory of dedicated core

asset developers. The project teams are now genuine business teams that can access and inform but not direct the execution teams. The focus of the product teams must change from product engineering to product advocacy with the stated goals of increasing product utilization by understanding customer needs that are not influenced by engineering constraints. By separating the business need from the engineering activities, product teams are genuinely focused on the actual customer. Tradeoffs between user desires and delivered products are handled internally to the organization and no longer conceded to by the customer. Shifting the product team's focus towards the customer and away from the factory team is encouraged by measuring performance of the product team, not by the number of fixed of features deployed but instead on non-technical metrics such as the number of training exercises executed or the number of enhancements request that grow system capabilities.

None of this can happen if the value proposition is not substantiated by the actual cost of system deployment for equal capabilities being lessened, or equal cost investments providing greater system capabilities. Technical performance measures must be maintained by the executing teams to ensure that the product teams' critical needs are met by the factory and that the execution of the factory is more efficient than a smaller dedicated team. Increasing efficiency includes ensuring that core skill sets are leveraged across the entire factory so that problems are addressed in the most effective resources. In aggregate, the use of dedicated experts that aren't pigeon-holed working on a specific project create substantial savings in time and rework while increasing the exposure to and knowledge of resources across the factory.

Leadership must maintain a unified schedule across all efforts for tracking work at the task level. Task selection must be evaluated with all products in mind. Shifting leadership resources between product teams prior to the transition to collaborative development reduces the impact after the transition to collaborative development. Switching of leadership resources can be made less painful by executing switches during times when product teams are executing efficiently and the subordinate teams have enough momentum to indoctrinate new leaders with minimal impact.

6.4 Use in LT2

Not all teams have successfully migrated to collaborative development in CPM. Smaller teams have been more effective at the transition because the limited number of resources available to them has mandated a holistic approach to the development and support of core assets. Larger functional groups tend to be the primary expense, and therefore focus, of product team management. Because larger teams are able to keep resources dedicated to a single team they are more directly associated with the identity of a specific product and (in our experience) the hardest to wean off that product-focused mentality. This identity association results in trends among developers to continue their previous associations and assumption of direction from product team leadership instead of factory lane leadership. Activities as innocuous as going to lunch strengthen ties and foster product-centric discussions.

Calculation of performance metrics performance has encouraged collaborative development efforts. Keeping focus on execution metrics that relate to specific short term goals such as delivering on time and maintaining a high quality baseline are retained from previous execution methodologies. Figure 8 shows the metrics calculations for products providing collaborative development of baseline fixes back to the shared repository. As products release from the common baseline, the focus on constituent makeup of the fixes indicates the degree of change that contributed to the quality of each product by the contributions of others. New metrics that measure performance against the conventional execution methodologies have been successful in the past and are good for conveying easy to understand cost avoidance within the product line. The LT2 product line is now also measuring performance against the ideal product line to generate a single massive product that comprises all features available.

This measurement encourages elimination of redundancy by quantifying the cost reduction associated with eliminating duplication. This approach is particularly appealing within LT2 because products are not "purchased." Feature selection is based on end user needs that vary depending on the size of the unit being trained, environment, unit type, and training duration. In this environment the primary drivers for feature selection are reducing development and sustainment costs while maximizing training capabilities and effectiveness.

7. Conclusion

Each product line execution methodology that has been revealed throughout the history of the LT2 product line has shown itself to be effective at delivering high quality training products to the end user. Empowering product teams to drive to a more effective state of product line engineering can only be accomplished when product teams are motivated to change. Figure 11 summarizes the methodologies and the enablers for evolving to the next level. By charactering the methodologies identified in this paper and using concrete examples that are clearly articulated and familiar to product teams CPM has been able to build awareness and formalize the process of improvement. Speed of migration to the product line is increased and what took LT2 10 years to accomplish for software is taking 1 to 2 years for other functional areas using this approach. Newly indoctrinated products into the product line are being easily assimilated, and products that are left untended during periods of inactivity are able to restart immediately without having to catch up to the latest baseline.

Cost avoidance data shows that LT2 is on the right track. This continuing transformation through the product line execution methodologies discussed in this paper has generated a significant return on investment to date within PM TRADE's live training system acquisition portfolio. The early approaches after LT2 was launched as a product line generated over $300 million in cost avoidance across the development of live training systems that include Combat Training Centers Instrumentation Systems, Home Station Instrumentation Systems, Instrumented Ranges, and Targetry. The approaches put in place under the purview of CPM are projected to save another $200 million over the next 2-5 years[3] [3].

[3] These figures are based on industry standard estimates of code cost, and are calculated assuming that post-deployment software support constitutes 70% of development cost and a life expectancy of 10 years. See [5] for a more detailed explanation.

	Isolation	Awareness	Shared Delivery	Shared Baseline	Collaborative Development
Characteristics	Independent teams with discrete goals. Aligns well to government contracting paradigm.	Communication between teams. Sharing of support artifacts. Loaning resources across product teams.	Common technology shelf. Shared access to source and supporting artifacts.	Single baseline shared by product teams. Consolidated CM ensuring baseline quality. Feature based product actuation.	Single team implementing multiple product inputs. Pro-active prioritization and agreement across products.
Benefits	Freedom from external influence. Low up-front investment. No reliance on others for individual success.	Standardization of interfaces, and data types. Alignment of high level decisions. No reliance on others for individual success.	Reduced initial development costs though reuse. Reduced duplicate capability development.	Immediate incorporation of changes across PL. Eliminates post development merging. Faster issue identification & resolution.	Increased sharing, reduced implementation. Product teams focus on customer needs not technical solutions
Enablers to Evolution	Management intervention to determine effort overlap. Fostered collaboration and sharing of resources. Common architecture and standard interfaces.	Management emphasizes capability reuse over new development. Focus development efforts on missing domain capabilities.	Establish single CM authority. Feature models to manage product variation. Merging divergent baselines.	Product teams separated from development teams. Product needs determined independent of implementation. Focus on overall (not product) cost reduction.	

Figure 11 Migration between product line methodologies.

8. Acknowledgements

The authors wish to thank Paul Clements and Charles Krueger from BigLever Software for their continued support and partnership on CPM in refining and improving the LT2 product line. The execution methodologies extracted though the analysis of LT2 product line evolution are largely influenced by their expertise in product line engineering practices.

9. References

[1] Bergey, J., Cohen, S., Donohoe, P., & Jones, L. Software Product Lines: Report of the 2010 U.S. Army Software Product Line Workshop (CMU/SEC-2010-TR-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

[2] BigLever Software, "BigLever Software Gears," http://www.biglever.com/solution/product.html

[3] Clements, P., Gregg, S., Krueger, C., Lanman, J., Rivera, J., Scharadin, R., Shepherd, J., Winkler, A. "Second Generation Product Line Engineering Takes Hold in the DoD," *Crosstalk – The Journal of Defense Software Engineering,* vol. 27, no. 1, January/February 2014.

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

[5] Dillon, M.; Rivera, J; Darbin R, Clinger, B. Maximizing U.S. Army Return on Investment Utilizing Software Product-Line Approach. I/ITSEC, 2012.

[6] Lanman, J., Becker, B., Samper, W. Joint Service Partnership: Extending the Live Training Transformation Product Line. 2009 Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, FL.

[7] Lanman, J., Darbin, R., Rivera, J., Clements, P., Krueger, C. The Challenges of Applying Service Orientation to the Army's Live Training Software Product Line. 2013 Software Product Line Conference (SPLC), Association for Computing Machinery (ACM), Tokyo, Japan.

[8] Lanman, J., Kemper, B. Second Generation Paradigm: By staying ahead of product growth, PEO STRI improves efficiencies. Army Journal for Acquisition, Logistics, & Technology (AL&T). Department of the Army, Pgs. 110-114. April 2012.

[9] Lanman, J., Kemper, B., Rivera, J., Krueger, C. Employing the Second Generation Software Product-line for Live Training Transformation. 2011 Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, FL.

[10] Rubin, J., Berger, T., Duszynski, S., Becker, M., Czarnecki, K. , Dubinsky, Y. "An Exploratory Study of Cloning in Industrial Software Product Lines, CSMR 2013.

[11] Toft, P., Coleman, D., & Ohta, J. "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," Donohoe, P., ed., *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1).* Denver, Colorado, August 28-31 2000. Boston, Ma.: Kluwer Academic Publishers, 2000: 111-132.

[12] U.S. Army PEO-STRI, "Live Training Transformation," http://www.peostri.army.mil/PM-TRADE/lt2_productline.jsp

[13] Van der Linden, F., Schmid, K., Rommes, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*, Springer, 2007.

[14] Weiss, D. M. & and Lai, C. T. R. *Software Product-Line Engineering: A Family-Based Software Development Process.* Reading, MA: Addison-Wesley, 1999.