Incremental Return on Incremental Investment: Engenio's Transition to Software Product Line Practice

William A. Hetrick
Engenio Storage Group - LSI Logic
3718 N. Rock Road
Wichita, KS 67226 USA
+1-316-636-8433
bill.hetrick@engenio.com

Charles W. Krueger
BigLever Software
10500 Laurel Hill Cove
Austin, TX 78730 USA
+1-512-426-2227
ckrueger@biglever.com

Joseph G. Moore
Engenio Storage Group - LSI Logic
3718 N. Rock Road
Wichita, KS 67226 USA
+1-316-636-8501
joe.moore@engenio.com

ABSTRACT
Engenio made the transition to software product line practice in order to keep pace with growing business demand for its products. By using an incremental transition strategy, Engenio avoided the typical upfront adoption barrier – the equivalent development effort of 2 to 3 standalone products – which in their case was projected to be 900 to 1350 developer-months. Engenio discovered that by making an upfront investment of only 4 developer-months, they were able to start a chain reaction in which the tactical and strategic *incremental returns* quickly outpaced the *incremental investments*, making the transition pay for itself.

Categories and Subject Descriptors
D.2.13 [**Software Engineering**]: Reusable Software – *domain engineering, reusable libraries, and reuse models*.

General Terms
Design, Economics, Management, Measurement, Theory.

Keywords
Software Product Lines, Transition to Software Product Line Practice. Incremental Methods.

1. INTRODUCTION
Engenio faced a daunting challenge. The business demand for its RAID storage server products was rapidly out pacing their ability to create, evolve and maintain the firmware for those products[1]. Software product line practice, which was just coming into view at the time, seemed to be the answer. But there was an apparent barrier. Prevalent in the software product line literature at the time was the assumption that it requires an upfront investment of 2 to 3 products worth of development effort in order to see return on those investments[2,3]. Based on metrics from two recent deployments, Engenio's projected upfront investment was 900 to 1350 developer-months of effort, with a development staff of 100. There was absolutely no slack in Engenio's contractually-dictated production schedules to divert this amount of upfront effort to re-analyze and re-architect controllers, re-engineer and componentize assets, redesign production infrastructure, re-define processes, and re-organize management and development teams.

Rather than attempt to surmount this formidable upfront adoption barrier, Engenio chose instead to make an incremental transition into product line practice. Not incremental in the sense of prolonging the time to make the large upfront investments and similarly prolonging the time to see returns. But rather, incremental in the sense of making a series of small investments, each of which would immediately yield the order-of-magnitude returns that are characteristic of software product line practice.

Engenio discovered that with a relatively small upfront investment of 4 developer-months for its first incremental investment, the cumulative returns began to quickly outpace the cumulative investments in terms of time, effort and money. As a result, the return on investment was almost immediate and the resulting "profits" in time, effort and money from the return on investment could be used to fuel the remaining incremental transition.

There are, of course, many different facets within a software development organization to consider when making an incremental transition to software product lines. For example, Clements and Northrop characterize 29 key *practice areas* that may be impacted by a software product line approach[2]. Any or all of these might be considered in an incremental transition. Engenio chose to incrementally address, in sequence, those facets that represented the primary inefficiencies and bottlenecks in their development organization. By eliminating the inefficiencies and bottlenecks in the most critical facet, the next most critical product line problem in the sequence was exposed and targeted for the next increment.

To date, the sequence of incremental steps in the Engenio transition to software product line practice can be characterized by the following four primary stages:

- Transition the infrastructure from conventional configuration management and builds to first class software product line variation management, configuration management and automated production

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

- Transition from team organization by products to team organization by core assets
- Transition from development processes defined by product releases to development process defined by feature releases
- Transition from validation and quality assurance for individual products to validation and quality assurance for all of the software product line assets

As of the writing of this paper, Engenio is in the fourth incremental stage.

2. BACKGROUND

In today's customer-driven environment, most companies target the needs of their prospective customers by creating a product line – a portfolio of closely related products with variations in features and functions – rather than just a single product. For companies that utilize standalone or embedded software in their products, this poses a serious problem. Tools and techniques for software development tend to focus on individual products. As a result, developing software for a product line is extremely complex due to multiple intertwined products, features and production deadlines – all aimed at moving target markets. These tactical software development challenges are big enough to impede the realization of business-critical goals and strategies. More specifically, they can greatly hinder a company's ability to hit market windows, provide competitive pricing, maximize product quality, and expand the scale or scope of their product line portfolio.

A new class of software development methods, tools and techniques – collectively referred to as *software product line development* – is emerging to address this problem, offering improvements in development time-to-market, cost, quality, and portfolio scale and scope[4]. What is most interesting is the *magnitude* of tactical and strategic improvements that are possible, not measured in percentage points, but more commonly in factors of two to ten. These improvements are so large that they impact the fundamentals of how companies compete[2].

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

The idea of manufacturing software from reusable parts has been around for decades, but success has been elusive. Recent advances in the software product line field have demonstrated that narrow and strategic application of these concepts can yield order of magnitude improvements in time-to-market, quality, portfolio scalability and software engineering cost. The result is often a discontinuous jump in competitive business advantage, similar to that seen when manufacturers adopt mass production and mass customization paradigms.

The characteristic that distinguishes software product lines from previous efforts is *predictive* versus *opportunistic* software reuse. Rather than put general software components into a library in hopes that opportunities for reuse will arise, software product lines only call for software artifacts to be created when reuse is predicted in one or more products in a well defined product line[4].

Conventional methods of portfolio development typically include ad hoc combinations of code-level techniques such as:

- configuration management branching or clone-and-own, which often lead to wasteful duplication and divergence of effort
- #ifdefs, templates, file naming conventions or runtime conditionals with configuration files, which often lead to unmanageable complexity

In contrast, software product line development tools, methods and techniques give first-class engineering status to the issues of portfolio development, providing constructs that allow development organizations to take advantage of the similarities among the products, to effectively manage the variations among the products. For example, see [5] for specific tools and methods utilized by Engenio in this case study.

- *Feature modeling* language, used by software architects to model the features that cause variation among the products in a portfolio and to model the feature profile for each of the products in the portfolio.
- *Variation points*, used by developers to encapsulate implementation-level differences among the products in the portfolio and the logic about how to instantiate a variation point based on the feature profile for a product. Variation points, which are use in software *assets* such as requirements, source code, test cases and documentation, enable a single software asset with variation points to be used for all products in a portfolio. This consolidation is what allows the portfolio to be developed as though it were a single system.
- *Configurator*, used by engineers doing a product build, takes a feature profile for a product and instantiates that product by composing assets and by instantiating variation points within those assets. The configurator enables *continuous builds* for the entire portfolio (analogous to continuous builds of individual products in agile methods), where any developer can automatically compose and configure all of the assets (requirements, source code, test cases, documentation) for any product at any time, based on the current state of the assets under development. For example, a developer's change to source code and test cases can immediately and automatically be built and tested in all products in the portfolio.

Software product line development approaches provide a shift in perspective, so that development organization can engineer their entire portfolio as though it were a single system rather than a multitude of products.

3. INITIAL STATE OF ENGENIO'S DEVELOPMENT PRACTICE

Engenio is in the business of providing feature-rich, high-performance, storage servers to major OEM vendors such as IBM, SGI, Cray, StorageTek and Teradata. Each OEM customer wants to take advantage of Engenio's core competence in storage technology, but each one also wants unique and differentiated solutions. Thus, the need for efficient product line engineering is central to its business model. Furthermore, Engenio has recently

completed the process of becoming a standalone division within its parent company, LSI Logic Corporation (Engenio was formerly known as LSI Logic Storage Systems), so the ability to extend its product line to attract and retain a growing customer base is key for establishing investor value.

Engenio's *Controller Firmware Development* group is the focal point for the evolution to a software product line. In recent years, this group has grown include to 180 developers, working at four distinct geographic sites. The group currently provides firmware for 82 products, with about one million lines of embedded software going into each product. Approximately 80% of the code is common among all products.

In the late 1990s, the Engenio Controller Firmware Development staff lived in far simpler world. The firmware was constructed as a single software build, applicable to all deployments. Any required product variability was resolved at runtime, using downloadable configuration data stored in non-volatile memory. Evolution of the firmware over time was managed using the concept of formal *releases*. These releases were managed in a disciplined manner, with a formal waterfall process guiding the way. Each release would transition through the stages of analysis, design, implementation and test. Once a release was complete and products were shipping to customers, work on the next release could begin. In retrospect, it is apparent that it was the sequential nature of the release cycle that kept the firmware development world simple and sane.

Market pressures eventually emerged that greatly disrupted the simple, sequential release-based perspective that had guided the firmware team through the nineties. The 21st century brought increased success in the marketplace, but this success was accompanied by a furor of development demands that could not be accommodated by sequential development cycles. The tranquil world of sequential releases was replaced with the nonstop chaos of a half dozen intertwined and overlapping release cycles.

In the midst of the market-driven turbulence, the need for change within the engineering group at Engenio was not readily apparent. Instead, the evolution from a single product to a product line stealthily crept into Engenio's product development world. The initial portent of change was the introduction of a new, low-end, hardware platform. While the feature content of the low-end product was to be the same and the bulk of the firmware code was to be shared between the new low-end and existing mid-range products, there were substantial differences between the two hardware platforms. The introduction of a second, substantially independent, development environment to support the new platform was the initial solution for managing this variability. Although this move solved the immediate problem of supporting the variation among the old and new products, it proved to be a less than ideal way to manage the commonality.

During the evolution from one to multiple products, the primary tool utilized by Engenio's firmware group was a proven, well-integrated, configuration and problem tracking system. This system did an excellent job in the role for which it was intended, but it created the tendency for product variability problems to be treated as configuration management problems. For example, variability to satisfy differing platform requirements was managed by branched versions of the same source files. Similar branching was introduced to support the concurrent development of diverse features. The end result of the evolution from a single product to a diversity of hardware platforms and firmware features was success in the marketplace, but chaos within the code base. Eventually, approximately 34% of the 3300 files in the Engenio source code base had anywhere from 2 to 16 active branches under development. The resultant branching, merging, duplication and divergence became the dominant effort in the firmware development team, leading to the realization that a change in development practice was imperative.

4. DECISION TO TRANSITION TO SOFTWARE PRODUCT LINE APPROACH

The negative impact of the rapidly increasing file branches became evident from two complimentary viewpoints, one from the architects and one from the managers. First, an architect began to observe code quality degeneration due to excessive file branching. As new features were implemented, there was a statistical likelihood that 34% of the impacted files would have branches (recall that 34% of the files in the code base were branched). The feature implementation on a branched file had to be duplicated on each branch of that file, which was labor intensive and highly error prone since each branch of a file might be slightly different in subtle ways.

Second, from the managers' viewpoint, defect resolutions were also becoming increasingly expensive, since defects resulted in a large percentage of development capacity spent on analysis, correction, and validation on multiple file branches. The change management tool in use modeled this duplication of defect resolution tasks as *clone* requests. The development management team then used this data to track *clone metrics* – the ratio of clone requests versus the total number of change requests. The management team considered clone work a waste item in the development capacity because the duplication of effort prevents a developer from working on new revenue-generating features.

Clone metrics were increasing over time, mirroring the code quality issue observed by the architect. The root cause was attributed to the file branching, which was a result of the growing number products under concurrent development, which in turn was a direct result of Engenio's success in their OEM business.

To the architects and managers faced with this growing problem, *software product lines* intuitively resonated as a viable solution, though no one on the team had direct experience with this emerging field of practice. Before launching into a full-scale change initiative, they decided to expand their understanding of basic product family development principles with consulting and an assessment by an outside organization specializing in software product line development tools and practices, BigLever Software[5].

During the assessment, a pilot project plan emerged based on a vision of the development and production environment after a full-scale transition. The development and production environment would be based a unified set of core assets, residing on the single common trunk in configuration management – no more branches. The software product line development tool, BigLever Software Gears[5], would be used to manage variation at the *source configuration binding time*. Product instantiation, or production, would be fully automated, driven from decisions in

the Gears *feature model* and extracted from the core assets. All of the common and varying source code would exist as core assets with *variation points* under Gears, so no manual *application engineering* would be required.

Over the next few months, the pilot project plan was refined. Two products were selected to become the initial baseline for core assets in the product line. The pilot was designed to be an initial transition step in a deployment, if proven successful.

The pilot project was budgeted as a six-week effort, beginning on May 17, 2004. After consolidating the two product code sets with the Gears feature model and variation points, the methods and results of the prototype production line were shared with all of the development managers, architects, and their superiors on June 8, 2004 – three weeks ahead of schedule. With considerable discussion and analysis, projecting business requirements and scenarios in the development organization, the group concurred that the product line environment would eliminate the branching and clone problem, enabling the development staff better satisfy Engenio's business and engineering requirements. Development management agreed to the final deployment plan on June 17, 2004, one month after the beginning of the pilot. After *hardening* the pilot prototype and documenting the new configuration management and production procedures, the new software product line development environment was put into daily production use by the development staff on July 30, 2004. The entire effort to get to this point was 4 developer-months.

5. ENGENIO'S TRANSITION

Engenio structured its transition to software product line practice into four sequential and incremental stages. Each of these four stages had nested incremental substructure. The objective was to make small incremental investments in product line practice – small enough not to disrupt ongoing production schedules – in an effort to immediately reap larger incremental returns on those incremental investments.

The four stages were ordered sequentially to address, in order, the most prominent inefficiencies and bottlenecks first.

- Transition the infrastructure and core assets
- Transition the team organization
- Transition the development processes
- Transition the validation and quality assurance

5.1 Transition of the Infrastructure and Source Code Assets

The most critical issue identified to address first was the growing problem of file branching degrading quality and productivity. To accomplish this, the development infrastructure was enhanced to support a single consolidated collection of core assets rather than highly branched files in configuration management. The subtasks within this stage were to:

- validate the approach with a pilot project
- create the software product line development and production infrastructure
- incrementally extract a core asset base from existing source code branches
- incrementally transition development teams away from branch and clone methods to core asset development using the new infrastructure
- incrementally refactor the core assets to maximize commonality and to optimize abstraction in variation points

When the transition started, eight product releases were concurrently in various stages of active development. The pilot project consolidated the branched source code for two of these products, resulting in the initial core asset base. To minimized the potential disruption of the critical product releases under development, the pilot and initial deployment strategically targeted two products that were early in their release cycle.

The initial code base for the two products was comprised of 3300 files, 1000 of which had anywhere from 2 to 16 branches, organized into 212 directories. The core asset base after consolidation was comprised of 3103 common files, 51 Gears variation point files, 211 common directories, 1 Gears variation point directory, and 0 branches.

The transition plan dictated that the remaining product releases currently in flight would be completed using existing branch/merge/clone methods. All subsequent product and feature releases were to deploy from the core assets of the product line. As illustrated in Figure 1, over the subsequent 9 months, developers rolled off of projects developed under the legacy methods, while the core assets and production line evolved to initially deploy 23 products.

Early in this stage of the transition, the Gears variation points were unceremoniously extracted from the existing branched code base with no attempt to re-architect or re-engineer the core assets other than to encapsulate, characterize, and program variation points based on a simple Gears feature model. This allowed, for example, all 51 variation points to be extracted from the 3300 files, 1000 of which were branched, in less than 3 days during the pilot. However, over time, the variation points were incrementally revisited in order find opportunities for greater commonality and to find better product line abstractions.

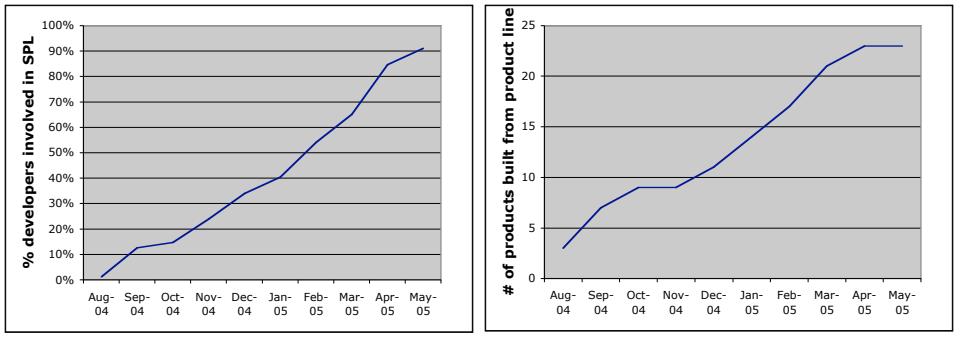

Figure 1. Incremental Transition of People and Products

The *incremental investment* during the first stage of the transition was to establish a new software product line development and production infrastructure, create a baseline of core assets from the existing branched source base, and train developers on the new Gears-based product line development environment and methodology. The initial investment to conduct the pilot project and then to go into live product line development and production with those 2 products was 4 developer-months. Subsequent transition of the remaining 21 products and product teams required less than one half developer-month per product.

The *incremental return on investment* originated from improved efficiencies at the tactical developer level. Developers spent less time manipulating the configuration management tool, navigating and making cloned modifications on intricate file-by-file branch structures. As a result they could spend more time developing new features.

The dramatic reduction in branch complexity is illustrated in Figure 2 using a *file branch factor* metric, defined as the number of file branches normalized by the number of products. Note from the graph that branches have been eliminated in the product line core asset base.

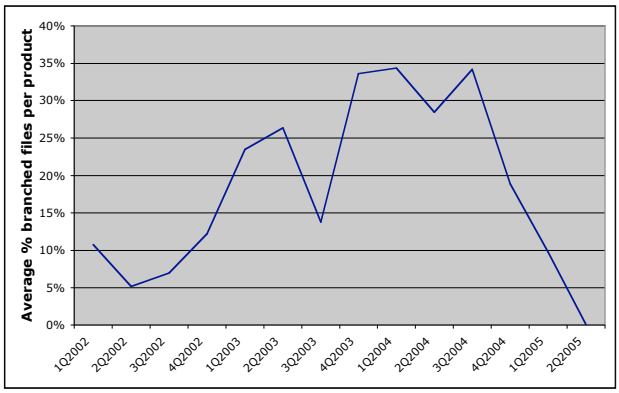

Figure 2. File Branch Factor During the Transition

Our historical metrics demonstrated a strong correlation between the number of branched files and the number of duplicate defect resolutions required. As a result, the reduction in file branches led to a derivative return on investment – the elimination of clone defect resolutions.

Finally, at a strategic level, Engenio quickly gained increased confidence in their software product line development capabilities and, as a result, their ability to respond to more aggressively to expanding business opportunities. As is typical with software product line transitions, the strategic returns on investment overshadow the tactical returns.

5.2 Transition of the Organization

The new software product line development environment and transition to core assets facilitated the development of concurrent product releases and also helped with the deployment and validation of new features across multiple hardware platforms. However, after the first stage of the transition, most of the development focus still remained product release centric. The second stage in the incremental transition was to start a shift in development focus to core assets by restructuring the development organization. Developers on product release teams incrementally transitioned to a set of teams defined by collections of core asset components. The collections of core assets were grouped by affinities and service layers in the firmware architecture.

Each team in the new organizational structure has a core asset manager and a technical team leader. The core asset manager is responsible for making sure that their core assets provide the right capabilities at the right time on the product release roadmap. The core asset technical team lead is responsible for implementing their core assets according to the architectural and feature requirements on the product roadmap. Core asset managers and team leads also have an internal focus with their team on *core asset stewardship*, or maintaining and optimizing the integrity of their core assets. Core asset stewardship includes tasks such as refactoring to optimize commonality and variability, maintaining an appropriate balance for the requirements tradeoffs across the entire product line, and searching for emerging abstractions in the variation points[6].

Asset team staffing was determined by existing and planned domain expertise needs of the clustered components and the asset evolution requirements for new features. This was a difficult transition culturally as developers, who were accustomed to working on any file in the code base, were now expected to request changes from the domain experts of other asset teams.

The investment for this increment was the planning to define the asset teams and training to educate team members, team leaders, and development managers in the roles and responsibilities of the asset team roles.

The return on that investment was better planning and a more controlled and efficient asset evolution. This has resulted in better designs and improved defect resolutions due to improved visibility into the product family and core asset requirements. Technical experts are making better generalized technical decisions for the assets they own, based on a deeper knowledge of those components. Having teams with ownership and deep expertise in a narrow collection of core assets ultimately reduced the rework in both designs and defect resolutions.

Although it has not been formally characterized yet, there is clear evidence that the feature development capacity within the organization has been increased by at least 50%. Rather than scrambling to reallocate overextending developers to meet production schedules, managers now have the luxury of choosing where to allocate this newly gained development capacity.

5.3 Transition of the Development Processes

One of the difficulties in transitioning the development team focus to an asset focus was that Engenio's development process was release centric. The third stage in the transition was to align the development process with product family development. The process needed to include a mapping step from feature and release requirements to asset requirements. It needed to increase communication effectiveness of product family requirements at the asset evolution level. The process needed to allow for an asset assembly and feature or product validation function through the development cycle. Engenio's process needs extended beyond compatibility with software product line practices by increasing the ability to respond to changing customer requirements as well as addressing the growing geographic distribution of the development staff.

Engenio made a significant investment to address its process needs by assembling a task force comprised of its Software Engineering Process Group (SEPG) along with management representation from other development and quality assurance roles. The task force held a face-to-face process summit with a clearly defined purpose of creating a software development process that addressed the needs of the company.

Amazingly enough, the group reached consensus at the summit on a process structure. The process definition was refined and incrementally put into practice. This stage of the transition completed the shift of asset team focus to product family development, further reducing the development overhead by satisfying multiple component requirements with coordinated development efforts and leveraging the domain expertise of asset team developers.

5.4 Transition of the Validation And Quality Assurance

The fourth incremental transition stage, currently in the deployment phase, is feature validation and quality assurance. This stage is improving the product engineering capability of the organization by completing all feature requirements validation iteratively as part of the development work, and shifting the responsibility of the product certification group to the interoperability matrix of Engenio's controller platforms, server systems, operating systems, network adapters, and network switches. This transition is another significant investment, affecting multiple cross functional groups and processes. The expected return from this investment is to move beyond efficiency improvements in development capacity to efficiency improvements in product production and release. For example, there are significant opportunities to reduce the time to complete final product validation, thereby further reducing the time-to-market for new products and new features.

5.5 Future Transition Work

The ultimate objective is to do software product line development at an optimal level of effectiveness and efficiency. After the four major transition stages are complete, we expect to continually evolve and search out areas of waste and overhead, though we expect these to be smaller and more focused optimizations rather than major transition efforts.

6. CONCLUSIONS AND LESSONS LEARNED

Engenio's transition to software product line practice demonstrated that a large development organization with a large legacy code base can make the transition to software product line practice without a large upfront investment and without disrupting ongoing production schedules. By investing only 4 developer-months of effort upfront and 12 developer-months overall, Engenio incrementally transitioned 23 products – each comprising 1 million lines of code – and 135 developers to a sophisticated software product line practice. This is two orders of magnitude less than comparable transitions reported elsewhere.

By staging an incremental transition, small *incremental investments* very quickly yielded much larger *incremental returns*. These returns – in effort, time and money – could then be partially or fully reinvested to fuel the next incremental steps in the transition. Furthermore, the efficiency and effectiveness of the development organization constantly improved throughout the transition, demonstrating that development organizations do not need to *take a hit* in order to reap the benefits of software product line practice.

Stated a little more strongly, the Engenio transition refutes the conventional wisdom that it takes an upfront transition effort equivalent to developing 2 or 3 standalone products in order to achieve return on investment. Engenio achieved return on investment after an incremental investment of only 4 developer-months of effort. In contrast, the conventional wisdom of 2 or 3 products predicted the return on investment for Engenio to be 900 to 1350 developer-months, or 200 to 300 times greater than that actually experienced.

The incremental transition approach provided the same strategic benefits as experienced by *waterfall* transitions. Following the initial transition of the infrastructure, organization and processes, the strategic benefits quickly materialized. Over the next 5 months, the size of the product line portfolio grew 225% – from 23 products to 52 products – expanding at an average rate of a new product every 4 days. At the time of this writing, the product line comprises 82 products.

A big part of this success, we believe, is that Engenio made an explicit decision to transition to software product line practice utilizing the emerging knowledge, technology and best practices coming out of the software product line field. The data and

experiences at Engenio are consistent with others that have utilized these emerging approaches[7,8].

Another of the lessons learned from this experience is that the transition in product line practice areas such as infrastructure, core assets, organizational structure, development processes, and QA practices do not have to occur in parallel. Engenio incrementally addressed these in sequential order, choosing to solve the problems with the largest potential return on investment first. With the sequential approach, after the transition in one area was complete, the need for transition in the next was more clearly illuminated. For example, after creating the production infrastructure and formal core assets, it was easier to argue why teams should be organized around core assets and limited to modifying their core assets rather than be organized around products and free to modify any source file.

The "people issues" are always the most difficult when imparting change in an organization[9]. A final lesson learned that we found to be particularly important was that the incremental transition made one aspect of the people issues easier than expected. By quickly and continually showing benefit, it is much easier to quell the detractors. In fact, we found that the biggest skeptics and detractors quickly became the strongest advocates, once they experienced the benefits of software product line practice. By quickly and incrementally showing return on investment, detractors have less time and opportunity to derail the initiative.

7. REFERENCES

[1] Engenio Information Technologies, Inc., Milpitas, CA. http://www.engenio.com

[2] Paul Clements and Linda Northrop. 2001. Software Product Lines: Practice and Patterns, AddisonWesley, Reading, MA.

[3] Davis Weiss and Chi Tau Robert Lai. 1999. Software Product-line Engineering. Addison-Wesley, Reading, MA.

[4] SoftwareProductLines.com, see Resources section, http://www.SoftwareProductLines.com

[5] BigLever Software, Inc. Austin, TX. http://www.biglever.com

[6] Charles Krueger and Dale Churchett. Eliciting Abstractions from a Software Product Line, in Proceedings of the OOPSLA 2002 PLEES International Workshop on Product Line Engineering. Seattle, Washington. November 2002, pages 43-48

[7] Ross Buhrdorf, Dale Churchett, Charles Krueger. Salion's Experience with a Reactive Software Product Line Approach. Proceeding of the 5th International Workshop on Product Family Engineering. Nov 2003. Siena, Italy. Springer-Verlag LNCS 3014, p 315.

[8] Charles Krueger. Variation Management for Software Product Lines, Proceeding of the Software Product Lines 2nd International Conference, SPLC 2, San Diego, CA, Aug 2002, Springer-Verlag LNCS 2379, p 257.

[9] John Kotter. 2002. The Heart of Change, Harvard Business School Press. Cambridge, MA.