Salion's Experience with a *Reactive* Software Product Line Approach

Ross Buhrdorf
Dale Churchett
Salion, Inc., 720 Brazos St., Ste. 700
Austin TX 78701 USA
ross.buhrdorf@salion.com
dale.churchett@salion.com

Charles W. Krueger
BigLever Software, Inc., 10500 Laurel Hill Cove
Austin TX 78730 USA
ckrueger@biglever.com

Abstract. Using a *reactive* software product line approach, Salion made the transition to software product lines with 2 person-months of effort, the equivalent of 1% of the effort required to build its baseline enterprise software product. This is two orders-of-magnitude less than the effort typically reported with *proactive* software product line transition efforts. Since the transition, Salion has achieved 90-day time-to-market intervals for seven new commercial products in its software product line, with each new product deployment requiring only 10% of the effort required to build its baseline enterprise software product. This report summarizes Salion's experience with a reactive software product line approach, some of the unique contributions of the experience, and generalizations that characterize how other organizations may benefit from reactive product line techniques.

1 Introduction

The Salion product line story offers insights into the opportunities and benefits of the *reactive* approach to software product lines as well as to the conditions under which a reactive approach is suitable. In this report, we first discuss the specifics of the Salion experience along a timeline: (1) the *start-state* prior to adopting a software product line approach, (2) the *transition* to a reactive software product line approach, and (3) the *steady-state* where products are deployed and maintained using a reactive software product line approach. Subsequently we draw generalizations from the experience to characterize ways in which other organizations might benefit from reactive software product line techniques. Additional details on the Salion product line experience can be found in an SEI case study[3].

The distinction between a *reactive* approach and a *proactive* approach to software product lines is in how the scope of the product line implementation is managed. In a proactive approach, all product line variations on the foreseeable horizon are implemented, in advance of when they may actually be needed in a deployed product instance. In a reactive approach, only those product line variations needed in current products are implemented[1][2].

2 Start State

Salion is a software startup, creating a new product for a new market[4]. As such, there was no prior experience building commercial systems in the domain.

The marketing and engineering teams started by analyzing, designing and implementing a generic baseline system intended to serve the target customer base. Software engineering techniques such as use case analysis, architectural focus, and modular design and implementation were utilized, though initially no software product line techniques were employed.

As Salion approached the market with the new product concept and baseline product, the need for a product line – as opposed to a single general purpose product – became clear. The need was identified for two binding times of variation, product build time (referred to as product *customization*) and product startup/runtime (referred to as product *configuration*).

3 Transition

Being a startup, Salion could not afford the upfront time, cost, and effort typically associated with adopting a proactive software product line approach. The baseline product required 190 engineer-months over the course of 12 months. If we extrapolate using the typical effort data described in the side bar *It Takes Two* on page 226 in [3], a traditional proactive transition would have taken Salion 2 to 3 times the effort of the baseline product, or 380-570 engineer-months over 24-36 months. This time and effort would have doomed the company.

Two key decisions characterize Salion’s transition to a reactive product line approach:

- The unmodified architecture, design, and source modules from the baseline product served as the core assets for the product line. Initially, no variations were introduced into the core assets.
- Tools and techniques to support product variation were put in place in order to efficiently react to new requirements for product variants. BigLever Software GEARS, an off-the-shelf software product line technology, was utilized to extend Salion’s existing configuration management and build system to allow multiple product variations to be created from its core assets[5]. Details of the product line approach adopted by Salion can be found in [8] and [3].

By utilizing the existing baseline product for core assets and off-the-shelf technology for software product line management and automation, Salion made the transition to a software product line approach in 2 person-months of planning and implementation effort.

4 Steady State

In the steady state, the Salion core asset base undergoes constant evolution in rapid reaction to new requirements. Requirements for change come from two sources: (1) requirements from new customers that cannot be satisfied from the existing product line, and (2) new internal marketing requirements to extend or refine existing capabilities of the product line.

Using this reactive approach, Salion has maintained 90-day time-to-market deployments for new customers. The total effort to implement these new product variants ranges from 5% to 10% of the effort required for the baseline product (i.e., factor of 10 to 20 productivity improvement)[9].

In order to maintain high levels of efficiency, Salion utilizes a GEARS *software mass customization production line* to generate all products from the core assets. All products in the product line can always be automatically “manufactured”, or generated, from the evolving core asset base. Products are never directly modified after being generated by the production line, so there is never any wasted effort on divergence, duplication, merge forward, or merge backward (see pages 41-43 in [8]).

Over time, Salion has gained experience on optimizing the balance between “thinking ahead, but not too far ahead” in its reactive approach. Being excessively proactive has proved to be wasteful due to inaccurate guesses about what customers might want. In contrast, much less time is spent reengineering when specific variants are implemented reactively when they are needed and when abstractions are refactored as they emerge from the evolving product line. An unexpected result is that the total source code line count of the core asset base continues to shrink even as new products, features, and variants are added to the product line[6][7].

The following two subsections describe two reactive scenarios, the first for new customer requirements and the second for new internal marketing requirements.

4.1 Reacting to New Customer Requirements

The security system in place at Salion is divided into two subsystems: Authentication and Authorization. The authentication subsystem is a common component and therefore is out of scope of any customization requirements. The authorization system is also a common component, but the rules that dictate who can do what in the system is configurable; the rules of which reside in an XML document.

Due to the complicated nature of the authorization rules, and the inherent cost of testing every permutation of user roles in the system, it was decided not to customize this area of the product.

After two years of using a stable and understood set of authorization rules, a new customer required a complex modification involving field-level security on the input screens and the addition of a group permissions mechanism. The latter requirement seemed reusable across all product instances, and was easily turned into a common feature. The daunting task of customizing the authorization rules remained.

Using a reactive approach and GEARS, the file containing the authorization rules was converted into a variation point, allowing the previously existing rules to be used by previously existing product instances, while the new rules were implemented and applied only for the new customer. The effort required to define the new rules was minimal and no common code was touched because (1) the framework mechanism was well-designed (it did not know or care about the rules contained in the XML definition file) and (2) the framework simply expected the file to exist (no coding tricks were required in order to ensure the correct XML definition file was loaded per customer).

A combination of the component based architecture, smart deployment strategy and early plans to mitigate such risks by installing GEARS allowed Salion to react quickly to a customer request that might have otherwise stressed the product line architecture.

4.2 Reacting to Internal Marketing Requirements

Salion's *Supplier Customer Relationship Management Solutions*™, an enterprise software system for companies whose revenue stream starts by receiving and bidding on *requests for quotes* (RFQs), consist of three modules: Opportunity Manager, Proposal Manager and Knowledge Manager. Opportunity Manager deals with the pre-quote sales and sales process management activities. Proposal Manager deals with the process and collaboration around the generation of the quote. Knowledge Manager deals with the analysis of the data that is gathered in the course of using the product suite.

In the initial design and development of the Knowledge Manager module, Salion knew there would be a need for summary “reports”. The challenge with creating a new product category is to focus on the vision of the critical parts of your solution and not spend too much time on solutions for the many non-critical parts. Reports were one of these non-critical areas that needed a good answer on a partially flushed out concept.

The initial attempt at defining the reports system proved to be too proactive, attempting to define in great detail all possible requirements that might be reasonable for the target customers. After 4 months of specification, analysis, and preliminary design on a reporting toolkit, it became clear that there was considerable effort, uncertainty, and risk associated with implementing the reports feature. At this point, we stopped and asked, “Is this where we should be spending so much of our time at this stage of the product?”

It was this uncertainty and the necessity to use resources wisely that led us back to a more reactive product line approach. Simply put, given that we were unsure about the long-term vision for reports, we chose to implement only the reports required to support the current customers, developing each customer-specific report as a product line variant. After implementing the reports for a few customers, abstractions have begun to emerge that can be refactored into the common core assets of the product line. This was the first of many areas where we utilized the reactive approach to put off the long-term definition efforts until experience with customers gave us accurate requirements.

Having the ability to focus on the critical functional design points and not on the non-critical areas has been a great benefit for the product and the customer. No time is wasted on proactive product definition in non-critical areas for which there are no clear customer requirements. Being reactive, we get it right the first time with each customer until we discover the right combination of features and functions for a given product area. The efficiency and effectiveness of our reactive product line approach gives us a competitive advantage over other enterprise class software companies.

5 Generalizing Lessons Learned for Reactive Approach

We believe that many of the lessons learned in the Salion experience with a reactive software product line approach can be generalized and broadly applied to other product line initiatives. In the following paragraphs we group the generalized lessons learned according to the timeline used earlier in this report (start state, transition, steady state).

Generalizations for the start state:

- If the proactive approach represents an adoption barrier due to the associated risks and investments in time and effort, then a reactive approach offers a lower risk and lower cost alternative.

Generalizations for the transition:

- If one or several existing systems have a good representative architecture, design, and implementation, then they can be used with almost no effort to create the baseline of the core assets for the product line (and the initial product), even if they were not created with product lines in mind.
- Off-the-shelf software product line tools and technology reduce the upfront time and transition effort.

Generalizations for the steady state:

- If at all possible, avoid divergence and duplication among products and core assets by engineering and generating every product build from the core asset base. Avoid further development on products generated from the core assets.
- If you can accomplish the previous bullet, then don't make a hard distinction between domain engineering and application engineering. Everyone should be working on the core assets and the production line that generates all products.
- Use customization and refactoring to find emerging abstractions in order to avoid inaccurate and over-generalized proactive design efforts.
- Initially use reactive customization for features that are not fully understood or that are low priority. A more proactive design and implementation can be effectively deferred until sufficient domain knowledge and time is available.

6 Conclusions

By broadly applying reactive techniques, we believe that software product line initiatives can be easier to launch, offer greater benefit, are more accurate, more agile, and more efficient, plus introduce less risk.

References

[1] Krueger, C. *Easing the Transition to Software Mass Customization*. Proceedings of the 4th International Workshop on Product Family Engineering. October 2001. Bilbao, Spain. Springer-Verlag, New York, NY.

[2] Clements, P. and Krueger, C., *Being Proactive Pays Off / Eliminating the Adoption Barrier*. IEEE Software, Special Issue of Software Product Lines. July/August 2002, pages 28-31.

[3] Clements, P. and Northrop, L., *Salion, Inc.: A Software Product Line Case Study*, Software Engineering Institute (SEI) Technical Report CMU/SEI-2002-TR-038, Carnegie Mellon University, Pittsburgh, PA, November 2002.

[4] Salion, Inc. Austin, TX. www.salion.com

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

[6] Krueger, C. and Churchett, D., *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] Buhrdorf, R. and Churchett, D., *The Salion Development Approach: Post Iteration Inspections for Refactoring (PIIR)*, Rational Edge, www.therationaledge.com/content/mar_02/m_salionDevelopment_rb.jsp, March 2002.

[8] Krueger, C., *Variation Management for Software Product Lines*, in Proceedings of Second International Conference, SPLC 2, San Diego, CA, August 2002, pages 257-271.

[9] Krueger, C. *Data from Salion's Software Product Line Initiative*, Technical Report 2002-07-08-1, BigLever Software, Austin, TX, July 2002, available as http://www.biglever.com/papers/SalionData.pdf