Software Product Lines and Configurable Product Bases in Business Applications – A Case from Financial Services

Paul D. Witman
California Lutheran University
witman@ieee.org

Abstract

Software Product Lines (SPLs) provide a methodology for creating products based on a common set of core assets. In an environment that is already broadly successful with software reuse, including large-grained software reuse, can Software Product Lines provide a framework to understand the successes of the organization? This paper presents a case study of a large financial institution’s internal software development group, and its use of SPL methods and maturities. Adoption of SPL methods in business applications and the organization to support them is examined. Two different SPL maturity levels are observed, and the organization has made explicit choices about maturity levels vs. time to market requirements.

1. Introduction and Motivation

While there has been much research on software reuse at the fine- and medium-grained levels, there is relatively little published on actual case studies of large-grained software reuse within corporate environments. This study focuses on a very large diversified financial services corporation, with significant software development capabilities, and their experience with large-grained reuse. It further examines how the organization has used and benefitted from Software Product Line (SPL) methods and maturities in its development practices. The organization has a long history of successful reuse at various levels of granularity, and recently was required to begin to build and manage an additional variation to their product line.

We have previously defined large-grained software reuse as object-code reuse of entire applications and systems, across environments with differing requirements for business rules, languages, presentation styles, and other elements [1]. This is fundamentally similar to Bosch’s “configurable product base” (CPB), the highest defined level of SPL maturity [2]. The varying product requirements are accommodated via business rules and other runtime-binding configuration tools that enable the same object code to behave differently depending on how it is configured.

Bosch views CPBs as a higher level of maturity than SPLs. As such, it would seem unlikely that an organization would deliberately move from a CPB to a SPL maturity level. However, there is evidence from this organization that the two maturity levels may in fact coexist for the same product line, allowing the benefits of both.

Much work has been done on software reuse in fine and medium granularities – objects, subroutines, etc. Relatively little work on very large-grained reuse of business applications within an organization is visible in the literature – for example, reusing an Internet banking application set for independent business units in North America, Europe, and Asia – all with relatively little customization and rewriting as compared to commercial systems.

BigFinancial (not its real name), and the BigFinancial Technology Center (BTC) in particular, have created a large number of software and hardware systems (large building blocks) that have been reused in multiple businesses and countries. More recently, BTC has created a product line from the core assets of its Internet banking platform and applications. BigFinancial and BTC thus provide a rich source of data for case studies to examine the characteristics of those products, such as system infrastructure, automated teller machines, and Internet banking systems, and why they’ve been successful.

BTC is a technology development unit of BigFinancial, operating primarily in the western United States. Approximately 500 people are employed by BTC, and it is organized to deliver both products and platform components to BigFinancial. BTC’s product line includes system platforms, consumer Internet banking systems, teller systems, and system management tools. BTC has previously demonstrated itself to be very successful in building many of these products as CPBs. It supports unique requirements
2. Literature Review

A “Software Product Line” approach [3-5] suggests that software components can be treated similarly to artifacts of physical manufacturing – reusable parts that contribute to consistency across a product line as well as to improved efficiencies in manufacturing. Benefits of such reuse include the high levels of commonality of such features as user interfaces [6], which increases switching costs and customer loyalty in some domains. This can logically extend to banking systems in the form of common functionality and user interfaces across systems within a business, and across business units.

Clements and Northrop define a software product line as a “set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [3]. They then identify the three essential activities for a successful SPL approach, including core asset development, product line development, and management.

Bosch [2] defines a range of software product line maturity levels. The main stream of these maturity levels flows as follows:

- Independent products
- Standard infrastructure - operating system, application server, etc.
- Platform - capturing all common functionality
- Software product line - as defined by Clements and Northrop [3]
- Configurable product base – large-grained reuse with configurable behavior

In addition, Bosch identifies two hybrid maturity levels. These are a Program of Product Lines, where-in multiple product lines are assembled into a program of product lines, supporting a wide range of product families for very large systems, and Product Populations, where a single SPL can cover more than a single domain of products.

Bosch [2] further identifies four organizational models in organizations using an SPL approach. These include:

- Development department – no specialization
- Business unit development – application engineering specialized by product
- Domain Engineering – separate domain engineering for one or more products
- Hierarchical domain engineering – hierarchy of domain engineering groups, supporting platform and product function core assets of a SPL.

Moving from a traditional model to one supporting software product lines requires a transition process. Clements et al [7] document an Adoption Factory pattern, focusing attention on product, process, and organization, and defining three phases of adoption including establishment of context, establishment of production capability, and operation of the product line. Key to the initial adoption are behaviors such as mining existing assets to create the core assets from which products may be developed.

Mohan and Ramesh [8] provide insights on change management patterns in software product lines. They note three major patterns of issues in change management, including interdependencies among changes in separate variants and reinvented variations. They also recommend solutions to these issues, including modularized changes and variation points, careful tracking of variations, and knowledge sharing to facilitate reuse.

Various authors have touched on software reuse and Software Product Lines in banking technology. For example, Banker [9] evaluated the impact of software reuse on development productivity in a banking organization, finding significant improvements in productivity and increases in reuse as the organization matured. Griss [10, 11] looked at construction of banking components as part of a product line, and evaluated the feasibility (in an experimental setting) of this form of implementation. Gomaa and Webber [12] evaluated the extensibility of an automated teller machine SPL using a technique called variation points. However, there appears to be little published work that directly addresses the actual use of SPL methods in banking and banking technology organizations.

Software Product Line practices seem at this time to be concentrated in embedded systems environments. Numerous SPL case studies are published...
3. Methods

In evaluating the research questions and research subject, a case study appeared to be an appropriate methodology. Yin [15] describes three conditions to be evaluated in determining the type of methodology to be used. These include the type of research questions to be asked, the control that the researcher has or needs over subjects’ behavior, and the degree of focus on contemporary vs. historical events.

Key research questions for this study are of the “how” and “why” form. No control of subjects’ behavior was required to conduct the study, as the objective was to study what had happened relatively recently and what is happening — leading to a focus on contemporary events.

Yin [15] provides a set of three criteria that characterize situations appropriate for a case study. First, where there are more variables of interest than data points available; second, where there is a need to examine multiple sources of evidence, and third, where there is preexisting theory on which the case might be based. Multiple sources of evidence, including archival documents, interviews, and physical artifacts, were available and can contribute to the data collection required for this research. Due to the ability, within BTC, to replicate the case study process across multiple cases, and to gain greater robustness of the results, a multiple-case study was appropriate.

Interviews were conducted, and detailed product documents reviewed, between May, 2006 and June, 2007. Interviews were semi-structured, with initial questions provided to the interviewee in advance to allow for preparation. Additional questions were added during the interviews, where appropriate, to address topics raised in the discussion. Where permitted by the interviewee, interviews were recorded. Interview notes were published in detail to the interviewee within 48 hours of the interview, allowing for their review to improve validity and reliability of the data.

Key to the design of a rigorous case study was early attention to establishment and retention of validity and reliability [15]. Critical elements of the plan included construct, internal, and external validity, as well as reliability.

Construct validity was addressed by a series of steps, including the use of multiple sources of evidence, establishing and maintaining a chain of evidence, and asking key contacts at the research subject organization to review drafts of the research. Internal validity was addressed by conducting pattern-matching and explanation-building as the research progressed.

External validity in case research requires analytic generalization, rather than statistical generalization as is used in survey research [15]. Analytic generalization refers to the use of a theory derived from or enhanced during a case study being used as the foundation for study of new cases [16]. Replication logic was readily leveraged in this case, as multiple cases were used. The generalization problem in case studies is similar to that of the natural sciences, as in experiments — these results will have to be replicated with similar conditions to be demonstrated to be generalizable [17].

Reliability was dealt with via two mechanisms. These were the use of a detailed case study protocol and the development and maintenance of a case study database [15]. These tools provide the ability for another investigator to review the same raw material and come to similar conclusions.

4. BTC Case Study

4.1 History of software development and reuse

Prior work with BTC provided an initial model to explain their success with large-grained software reuse [1]. This model considered elements that might affect both organizational intent and organizational practices that enabled success. Since that initial review, additional data have come to light indicating the possibility of additional SPL practices contributing to the organization’s success.

Use of SPL practices may contribute to more effective management of large-grained reuse environments, via the discipline of variant definition and management, as well as via the explicit separation of application- from domain-engineering teams [2, 3]. BTC provides primarily domain engineering functions, while its regional technology partners provide primarily application engineering functions.

The authors conducted initial interviews in May and June, 2006, with BTC management to evaluate the potential cases to be used for the case studies. Approximately 20 cases were offered by BTC, across numerous product lines, and dating back as far as the mid-1990s. Three cases, out of the initial set of 20,
were reviewed in detail during those interviews, focusing on cases that establish the historical track record of BTC in creating and deploying reusable software at both medium and large granularities. From this initial set of data, the researcher evaluated the contribution each case could make to both theoretical and literal replications of results [15].

The researcher considered whether it would be useful to examine older cases (e.g., BTC’s work on ATM systems and early versions of Internet banking), or at least to document the older cases’ history at a high level, in addition to detailed study of more contemporary cases. However, those cases pre-dated any elements of SPL approaches, and thus were rejected for this study.

In addition to the interviews, high-level project summary documents were reviewed for each of the selected cases. These documents identified implementation locations and timeframes, along with software version information. Physical artifacts, in the form of Internet banking implementations viewable on the Internet, were also reviewed to further triangulate the data available from interviews and documents.

For the purposes of this study, two of the three key projects were further analyzed. The first of these, the Java Banking Toolkit (JBT), provided the foundation for large-grained reuse of numerous banking service applications across multiple business units. The second, Worldwide Single Signon (WSSO), supported federation of servers and services using a single user-visible credential set.

Initial findings indicated that several current and recent projects showed significant reuse across independent business units that could have made alternative technology development decisions. The results are summarized in Table 1.

It is worth noting that several business units used local solutions rather than reusing the shared components. Reasons for this vary, but the most commonly cited reason was an imperfect fit of the reusable application with local requirements. These requirements gaps were progressively narrowed over time, causing the numbers of businesses using local solutions to decrease. Regulatory issues and market requirements also impacted some products, such as ATM systems that required specific functionality not available in the shared software and hardware products. An example of this is cash accepting features commonly used in Japan.

<table>
<thead>
<tr>
<th>Project</th>
<th>Reused in</th>
<th>Business Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Infrastructure</td>
<td>Consumer Internet banking; Automated Teller Machines</td>
<td>All users of BTC’s legacy Internet banking components – &gt;35 businesses worldwide. (Three units used a local solution.)</td>
</tr>
<tr>
<td>Internet banking</td>
<td>Europe</td>
<td>&gt; 15 business units</td>
</tr>
<tr>
<td>Internet banking</td>
<td>Asia</td>
<td>&gt; 10 business units (One unit used a local solution.)</td>
</tr>
<tr>
<td>Internet banking</td>
<td>Latin America</td>
<td>&gt; 6 business units (Two units used a local solution.)</td>
</tr>
<tr>
<td>Internet banking</td>
<td>North America</td>
<td>&gt; 4 business units</td>
</tr>
</tbody>
</table>

Source: BigFinancial, BTC.

4.2 Interviewees and documents

BigFinancial and BTC provided access to a significant number of senior managers and key line employees to provide information about their relevant areas of expertise related to this study. The interviewees included:

- Director of Development for BTC – provided background on projects and overview of practices and organization at BTC.
- Director of Overseas Deployment for BTC, responsible for configuration and delivery of BTC’s products to its BigFinancial operations units primarily outside the US – provided insights into how BTC’s products were deployed, and how a single code base was maintained.
- Chief Architect for BTC, responsible for architecture of BTC’s product sets – provided technical details about BTC’s products and architectures.
- Lead engineer for BTC’s Worldwide SSO product – provided detailed insights into the evolution of the product over time, and the integration with the Java Banking Toolkit.
- Technology Head for BigFinancial’s lead deployment region for JBT – provided insights into the viability of the Java Banking
Toolkit to satisfy a wide range of business needs from the same object code.

- Director of United States Consumer Business Deployment for BTC – provided insight into the rationale for creating a separate JBT application code base, and beginning to operate more like a Software Product Line.

BTC provided access to all documentation for the chosen products, including business and technical requirements, design documents, test plans and reports, and various administrative documents, including project plans and meeting notes. Across the two products, over 1100 documents were provided and used as foundation to the interviews, and to provide triangulation on the information provided by the interviewees.

4.3 BTC Organization

Organizationally (see Figure 1), BTC is part of BigFinancial’s North American technology organization, with an indirect reporting relationship to the corporate technology organization. While geographically separate, it is placed as a peer to regional and local technology units, providing technology products and expertise to the regional and local technology and business units. It also acts to help deliver new functionality developed at regional business units to other regions, serving as a center of expertise for channeling functional enhancements between regions, for building globally reusable components, and for improving component quality and development processes. This commonality of experience encourages construction of global products, driving domain engineering up and application engineering down.

Despite BTC’s organizational peer relationship, it operates in some ways at a more corporate level than its regional technology partners. It is responsible for coordination of activities across the regional units, and is also responsible for the domain engineering function that builds core assets to meet the aggregated requirements of the regions. BTC’s central role in providing expertise to the regional units enables BigFinancial’s success in increasing the ratio of domain engineering to application engineering.

BTC’s primary development role is focused on domain engineering, as suggested by Bosch [2] for a CPB maturity level. There is also an element of a hierarchical domain engineering function at BigFinancial, with lesser amounts of domain engineering taking place in the regional technology units. BTC’s deployment groups and the regional technology units, in creating the configurations that control the behavior of the core assets, also perform application engineering functions to create products meeting the specific requirements for each deployment.

In recent years, BigFinancial has started to centralize technology activity into its regional technology organizations, and to remove technology units from individual lines of business. This has contributed to minimizing organizational barriers to reuse, as there are now fewer organizations with the technical expertise to propose and implement alternative offerings. As such, it simplifies the organizational negotiation around what solution to choose for a given technology problem. In addition, the business units’ growth of experience over the years with BTC, and with its product line and associated processes (software, financial, etc.) also contribute to the quality of the interaction model among the organizations.

There are signs of globalization across the regional technology units as well, with certain regional groups taking on inter-regional responsibility for specific areas of technology development and support. This promotes inter-regional commonality of technology choices, helping to ensure improved reusability of systems globally, and reducing overall costs.

Software development at BTC includes functions focusing on selection and integration of vendor and open-source components which will ultimately be used as core assets. In addition, the development group is responsible for the creation of software to extend and integrate those components, including tools to support language and rule configuration. These functions are all part of the domain engineering process, and given the late binding of variability of the products, the bulk of BTC’s engineering is classified as domain engineering.

BTC does perform some additional application engineering, in the form of applications built for a specific business unit, and collaborates with its regional technology partners to build some of those applications as well. BTC is thus performing application engineering, leveraging components from the core assets (infrastructure, platform, and domain components) to build applications specific to a business or regional need. BTC’s development group has responsibility for quality assurance and performance testing for the core assets, and is also responsible for field support for the regional technology units.

BTC is responsible for managing deployments to all regions. Within BTC, there are two organizational units for managing deployments, one focused on the US consumer banking market, and one focused on all non-US markets, and some specific lines of business in the US market. These BTC units are comprised of relationship and project managers who work with their counterparts in the regions and businesses to gather requirements, manage projects, and plan dep-
deployments, installations and support of the global products. The deployment groups are also responsible for creating the configuration packages that will govern the product behavior.

4.4 Java Banking Toolkit (JBT) and Applications

In the late 1990’s, BTC was responsible for creation of system infrastructure components, built on top of industry-standard commercial operating systems and components, to support the banking functionality required by its customers within BigFinancial. The functions of these infrastructure components included systems management, high-reliability logging processes, high-availability mechanisms, and other features not readily available in commercial products at the time that the components were created. The same infrastructure components were used to support consumer Internet banking as well as automated teller machines. The Internet banking services will be identified, for this study, as the Legacy Internet Banking product (LIB).

The requirements for the Java Banking Toolkit (JBT) as the successor to LIB called for several major functional elements. The requirements were broken out among the infrastructural elements, supporting the various planned application packages, and the applications themselves. The applications delivered with the initial release of JBT included a consumer Internet banking application set, an account activity and balance alerting function, and a portal toolset.

Each of these applications was designed to be reused intact in each business unit around the world, requiring only changes to business rules and language phrases that may be unique to a business. One of the fundamental requirements for each of the JBT applications was to include capabilities that were designed to be common to and shared by as many business units as possible, while allowing for all necessary business-specific variability, according to the requirements documents for the products.

Such variability was planned for in the requirements process, mining from the LIB infrastructure and applications, as well as the legacy portal and alerts services that were already in production. Examples of the region- and business-specific variability include language variations, compliance with local regulatory requirements, and functionality based on local and regional competitive requirements.

All core banking service functionality is supported by a single global application set. There remain, in some cases, region-specific functions required by a specific business or region. The JBT architecture allows for those region-specific applications to be developed by the regional technology unit as required, often leveraging core assets as the foundation.

An overview of the JBT architecture is shown in Figure 2. The core component of the architecture is a model-view-controller package, based on the open-source Struts and Tiles package. These components are extended to implement the domain requirements. Configuration settings for business rules, language choices, personalization, and the like are bound at runtime, based on settings in the configuration back- ing store. The platform connects to banking data sources (hosts) through a variety of custom connection components. Specific message routing decisions are made by a message broker, independent of the model logic, to separate business logic from data sources.

The United States (US) consumer business unit of BigFinancial made the decision in the early 2000s to separate its code base from the global code base for its account acquisition and marketing functions. These have since operated totally autonomously to the global portal services functions used in the other regions.

More recently, the US consumer business also chose to create its own version of the account servicing application components used by the other regions. According to the US Deployment Director, this was done based on the greater requirements imposed by the US consumer business, and the need to deliver more quickly than was possible with a configurable product base. This represented an implicit decision to move to a lower maturity level per Bosch’s framework, with the expectation of faster time to market for those components.

The existing global code base was mined as a starting point for creation of a US-specific application set. Some updates to the US-specific application set are returned to the global code base to be shared with other regions, while others are more unique to the US environment, and remain as application-specific components.

BTC manages this US variability with two separate application engineering teams responsible for the US application components, working alongside the domain engineering teams that produce the global applications and infrastructure. Two teams are used to provide sufficient capacity for the volume of new development required by the US business. These application engineering teams use components shared with the domain engineering teams, and may create variants of those components. They may also at times return the variants to the domain engineering team for integration into future global application releases. This appears to follow some of the patterns proposed by Mohan and Ramesh [8].
With numerous independent operators of the software systems delivered by BTC, managing revision levels is a challenge. Users with a critical bug or small feature request often want that feature delivered on top of their current version, which may not be the latest stable version available from BTC. As a result, variants sometimes must be created from older versions, with those content items added to the latest version at a future release.

4.5 Worldwide Single Signon (WSSO)

The Worldwide Single Signon (WSSO) functionality was conceived in the late 1990s. Early versions of online banking functionality were becoming more readily available, and additional Internet-facing features were being created by both internal groups at BigFinancial and by vendors and partners. BigFinancial’s various business units were attempting to offer a rich consumer experience on the web, but to do so they were required to integrate offerings from multiple sources, servers, and vendors. The WSSO product is delivered independently for reuse across numerous products within BigFinancial, and is also delivered as a core asset component of the JBT product.

The initial version of WSSO was built in 1999, and its first target was services for small business banking. This version leveraged a commercial encryption package to create a secure token that was encrypted and signed by the authenticating entity, and was verified and decrypted by the relying entity. Contents of the token included such data items as basic customer identification information, transaction time stamp, and other demographic data that could be used by the relying entity to establish a session for the end user.

Additional functionality was added to the initial package over the next few years, including components for session coordination, support for additional encryption models and token formats, and support for additional operating platforms. This latter addition was accomplished while porting the code to Java from the original C++. With each successive version, the product was delivered in object-code form to all customers, and made configurable and backwards-compatible, thus operating as a CPB.

BTC reports that WSSO is actively reused in a number of product lines within BigFinancial, and in cooperation with third party vendors and business partners. Products integrating WSSO included consumer and small business offerings, and ranged from portal services to credit card services, payment services, and alerting services. In addition to WSSO’s use within JBT (for portals, Internet banking, and alerts), it is also reused independently (e.g., for account opening, online statements, and brokerage services).

The requirements for WSSO have evolved through the years, both with additional usage within BigFinancial and among its partners, and as technology needs of the businesses have changed. As noted previously, the initial requirements focused on a single business, but the initial design supported extensibility for other environments and encryption models.

5. Observations and Limitations

5.1 Observations

While operating strictly on a code-reuse/single code base model, BigFinancial and BTC exhibit significant usage of SPL methods and practices as extended by Bosch [2], in building a configurable product base. The details of exactly how BTC develops and reuses its components were critical to that evaluation. It was found, after review of BTC’s activities, that BTC’s traditional (large-grained reuse) model does not directly match Clements and Northrop’s SPL concepts for its global applications. BTC’s practice was to modify only business rules and other configurations, rather than modifying parts of the product line components to provide features specific to a particular product incarnation. Thus BTC was found to be operating at a CPB rather than a SPL maturity level.

JBT operates as a configurable product base in many respects. The majority of the total functionality for each business unit is provided by a shared, globally reused code base, configured at run time. BigFinancial extends that model with region-specific applications that are often based on existing core assets, and are often returned to the global asset base for reuse regionally or globally. BTC’s mining of prior legacy products for core assets for reuse also appear to use at least part of the Adoption Factory pattern [7]. This enables both BTC and BigFinancial’s regional technology groups to more quickly build or configure applications for JBT to meet business needs. It is also worth noting that the SPL model appears to function effectively across wide geographic separation.

JBT’s US application set has recently been created as a separate product line, based on the core assets from the global application set, and appears to be managed as one application set derived from and built on the global application base. In these ways, BTC appears to be using SPL methods, in the evolution of variants from a common code base.

BTC’s development group operates primarily as a domain engineering function, while its deployment groups and the regional technology units operate as
application engineering functions. BigFinancial’s organization structure is unusual in that BTC is placed organizationally as a peer to its regional technology partners, but functions more at a corporate level to provide various technology leadership functions, including performance and quality assurance processes, global requirements leadership, and coordination across regional units.

BTC and BigFinancial made an explicit decision to move from a configurable product base to a software product line for the US version of its JBT product set. Based on its own observations about time to market for domain-engineered components, BTC determined that it would achieve faster time to market by using a software product line for its US customer. This is supported by the literature, noting that initial time to market for reusable components is often longer than for custom-built components [4].

This combination of CPB and SPL maturities for a single product line may be a new combination of maturity levels not fully accounted for by Bosch. Only the Program of Product Lines integrates CPB and SPL maturities [2]. BTC’s usage for its US and global Java Banking Toolkit applications does not match the model of multiple separate SPLs being combined into a larger SPL, as described by Bosch.

BTC’s WSSO product also operates as a CPB. WSSO fulfills a specific set of functions in a configurable way, and is reused intact across all products that integrate it. WSSO’s reuse does not come in the form of developing more instances from a common set of core assets. Rather, WSSO is itself reused intact, to support the needs of each of the various businesses in a highly configurable fashion. In that sense, given WSSO’s function as a component of JBT, it operates as a core asset for the Java Banking Toolkit. It is used in the construction of products on the JBT platform for those products that require Single Signon capabilities.

Like the JBT product, the WSSO product also tends to behave like a product line on a temporary basis as well. New feature requirements or bug fixes are required to be applied to an older version, creating a variant that is later merged back into the domain artifacts.

An additional instance of the SPL maturity level may well be used in applications being created for the credit card, mortgage, and consumer finance operations of BigFinancial. These groups may leverage the US model and mine from the core asset base of global applications, creating separate variants from those that are unique to their individual business needs.

BTC’s central role in BigFinancial clearly contributes to BigFinancial’s success in high-maturity SPL approaches. In this role, BTC is positioned to support domain analysis in what Neighbors called a “strategic planning role” [18], supporting aggregation of product line requirements across regions, and supporting the sharing of core assets for construction of applications. BTC’s history of successful reuse, and its long relationship with its regional technology partners, enables it to operate in a leadership role despite its peer relationship in the organization chart.

SPL methods were thus used by BTC to extend their capabilities to deliver products to customers. The SPL methods appear to operate in synergy with BTC’s existing large-grained reuse model. The SPL methods enable BTC to continue to build globally-reusable applications (as a configurable product base), while supporting a software product line model for its US customer.

5.2 Limitations

This study looked at only one institution, and focused on one primary development organization within that institution. This clearly limits the generalizability of the study. The study is limited to the banking industry, and to a multi-national banking organization, and so its findings may not be applicable to smaller institutions or to other industries.

While BTC’s work shows elements of the use of SPL methods and practices, neither BTC nor BigFinancial staff was focused on this stream of work. Therefore, inferences about what is actually happening are based on evidence provided by the research subjects, and not based on a demonstrated program of compliance to SPL methods and practices.

6. Conclusions and directions for future research

The two cases (JBT and WSSO) examined here have shown significant use of high-maturity SPL methods and practices. The organization has made an unusual choice to operate one of its product lines both as a configurable product base and as a software product line. This appears to be a new variation on the maturity levels defined by Bosch [2]. We would argue that organizations will make pragmatic decisions about software development models to best optimize resource and delivery requirements for their clients.

The organization is operating outside the area of embedded systems, and thus provides another source of evidence of the potential applicability of these methods in business application systems. Further, BTC and its BigFinancial technology partners are operating as internal technology suppliers, offering another variation from the more commonly-seen SPL adopters such as software or systems vendors. BigFi-
The financial’s organizational model provides examples of the separation of domain and application engineering functions.

Future research opportunities include the observation of ongoing change management at BTC, to validate that SPL methods and practices are indeed sustained, and to observe changes in maturity over time. BTC has now been introduced to the concepts of the Software Product Line and Configurable Product Base, and may choose to adopt these maturity levels more explicitly.

We will also seek out other application software vendors or internal development groups using this approach, to attempt to gather more evidence of SPL adoption in application systems. Numerous opportunities for such adoption exist, primarily in the licensed software environment, where large packages of core components are delivered to clients and customized to their specific needs.

In addition, the growing market for Software as a Service (SaaS) will provide opportunities for research on applications of Configurable Product Bases. SaaS vendors appear to use a configurable product base to support their high degree of scalability and rapid delivery.

7. Acknowledgements

The author is grateful to the research subjects for their candid input, and especially to Dr. Timo Käkölä and the reviewers for their constructive feedback.

8. References


Figure 1. BigFinancial/BTC Organization Structure

Figure 2. Java Banking Toolkit Architecture Overview