A make / buy / reuse feature development framework for product line evolution

Savolainen, Juha; Mannion, Mike

Published in:
Proceedings of the 20th International Conference on Engineering of Complex Computer Systems

DOI:
10.1109/ICECCS.2015.14

Publication date:
2014

Document Version
Peer reviewed version

A Make / Buy / Reuse Feature Development Framework for Product Line Evolution

Juha Savolainen
Danfoss Power Electronics A/S
Ulsnaes 16300 Denmark,
JuhaErik.Savolainen@danfoss.com

Mike Mannion
Glasgow Caledonian University
70 Cowcaddens Road, Glasgow
m.a.g.mannion@gcu.ac.uk

***Abstract*— In software product lines, feature development options can be categorized as make or buy or, a variation on buy, called reuse. In transaction economic cost theory a group of features is an asset and there is an increasing correlation between an asset's relative importance to a supplier and the decision to make rather than buy. In this paper we argue that the make or buy decision should also consider an asset's relative importance to the set of customers who buy the products containing the asset and we present a decision-making feature development framework that factors in feature relative importance to supplier and customer. To evaluate our framework we compared its recommendations with actual decisions made on three different industrial product lines. Our results showed broad consistency between framework guidance and actual practice, but revealed some instances not adequately explained by the framework.**

Keywords—feature, product line, make-buy

I. INTRODUCTION

A software product line business strategy focuses on commercial objectives and the development and evolution of supporting strategies to achieve the objectives e.g. marketing, sales, product development, and customer support. The continuous alignment of all these strategies depends on several contextual factors including demand patterns, product architecture, brand, culture, structure, governance, markets, competitors, geographical location, and recognition and reward systems. A common recurring technical challenge as the product portfolio and competitive landscape evolves over a long period of time is choosing the approach to construction for some particular group of product features. This paper provides a make/buy/reuse decision-making framework for constructing four different product feature types in the context of three different business strategies.

The discipline of transaction cost economics [1] provides a broad theoretical context to situate our contribution. A transaction is an economic exchange of an asset (goods or services). The decision to organize transactions within an organization or to outsource them on the open market—the *make* or *buy* decision—depends on the relative transaction cost values of each approach. Cost values are evaluated against a range of criteria including consideration of asset specificity to the supplier (e.g. specialized physical and human capital, firm-specific knowledge, operational capabilities), the amount of uncertainty about the future, third party capability and reliability, transaction frequency, the ready availability of technology to continue to access the asset, and the costs and benefits of various contracting alternatives. However Klein [2] noted from a range of empirical studies that *make* was more likely to occur the greater the asset specificity.

In this paper an asset is some subset of all the features in a software product line. Product feature development options are categorized as *make*, *buy* i.e. acquire-and-integrate-as-is, or a variation on *buy* called *reuse* i.e. acquire-and-integrate-by-adaptation where "acquire" can include commissioning by a third party, the purchase of a 3rd party product, the deployment of open source software or reusing from another part of the same organization. In practice all three options are often deployed for different groups of features. We argue that an asset's specificity evolves and that at any given time it should be judged not only by a consideration of its relative importance to a supplier but also by its relative importance to the set of customers who buy the products containing the asset, and we present a make/buy/reuse decision-making framework for different levels of specificity. Our Research Question was framed as: *What advice can be given to guide make or buy or reuse features given the differing relative importance of the feature to the customer and the supplier?*

In section III we explain how a product feature's relative importance is perceived from both a Customer's and Supplier's perspective and how such designations can change over time. In section IV we describe three broad Supplier business and engineering strategies and explain how this strategic context affects perceived importance. Section V sets out a make/buy/reuse decision framework and sections VI and VII explains how we used the framework to review a frequency converter product line infrastructure. Sections VIII and IX discuss the value and limitations of the framework.

II. RELATED WORK

Williamson [3] argues that minimization of transaction costs could be achieved by aligning transactions with governance structures i.e. markets or vertical integration within an organization or hybrids that combine some level of central coordination and protection for specific investments with market incentives e.g. organizations with highly decentralized assignments of decision rights, or franchises. Markets usually provide the greatest incentive structures to maximize profits, whilst vertical integration offers greater protection for specific investments and provides relatively efficient mechanisms for responding to change where coordinated adaptation is necessary, albeit with additional bureaucratic costs.

Bidwell [4] argued that a firm's ability to achieve transaction cost alignment is shaped by the interaction

between the structure of the organization and the nature of the decision problem. Often different organizational units focus on the problems of managing governance-related issues of cost and organizational flexibility on the one hand, and managing knowledge and the development process on the other. Such differentiation allows problems to be simplified to a level where individual decision makers can manage them but can create difficulties in solving problems that have simultaneous implications for the goals of different units, or problems that require information that is dispersed among different parts of the organization.

In psychophysics Weber-Feschner's Law [5] defines the concept of a just-noticeable difference (JND) i.e. the amount something must be changed in order for a difference to be noticeable. For suppliers the challenge is to understand how much they can or need to change their products, to surpass the JND threshold, in order for their customers to perceive a significant improvement. This will inform pricing charges.

In a software product line context this paper is a contribution to product line scoping. A product line scoping process consists of product portfolio scoping, domain scoping, and asset scoping [6]. Product portfolio scoping identifies products and features to be included in a product line. Domain scoping identifies the principal common functional areas of the product line domain. Asset scoping identifies specific assets with costs and benefits estimation from them. Different methods have been presented that cover different degrees of detail for each of these steps [7-11] and a comparison of these approaches, harmonized with the international standards (i.e., ISO/IEC 12207 and 15288) in both process and general terminologies is presented in [12].

At the product level the issues of make or buy have also been a focus for considering the merits of whether to deploy Commercial-Off-The-Shelf solutions. Most of this work has been presented in the context of developing a new system and whether at the outset to acquire one or more COTS. This choice is not easy, and in [13], the authors argue that COTS evaluators need to understand the impact of COTS software products on the system development process, determine evaluation requirements for COTS software, develop COTS software evaluation criteria, select COTS software evaluation techniques, employ a COTS software evaluation process that addresses the inherent trade-offs. In [14] a comprehensive overview is presented of many of the make-buy-commission decisions in product development including cost (direct costs, indirect costs, opportunity costs), capability (to make, to acquire, to manage commissioning, to support maintenance), scheduling constraints, staff availability, and the expected quality and fitness of purpose of the feature delivered that each alternative offers. Pohl et al [15] present an example of an engineering process for selecting a high-level COTS components during domain design (CoVARS – Component selection that considers Variability, Architecture and Requirements). It has three principle stages: component screening, detailed component evaluation and component selection. However it acknowledges that other non-technical factors such as alignment to business strategy, legal aspects and return on investment are outside its scope.

The focus of this paper is on the strategic business issues to consider when implementing features. So although we have assumed a compositional approach to implement variability, we are less concerned here with the method of architectural construction. For example, in delta modelling [16], diverse systems are represented by a designated core system and a set of deltas describing modifications to the core system. A particular product configuration is obtained by applying the changes specified in the applicable deltas to the core system. Typically an architectural delta can add, remove and modify components and alter the communication structure between these components. The decision to do so should be informed by the framework presented in this paper.

III. FEATURE DEVELOPMENT

A feature is an end-user visible characteristic of a system and their role in software product line variability modeling was first developed in Feature-Oriented Domain Analysis [17]. Most software product line engineering projects now include the significant task of identifying and describing the key features of each product in the product line [18]. The set of product descriptions is captured in a single feature model that contains all common and variant features of the software product line at different levels of abstraction. It can be helpful to organize a feature model as a forest, in which the features are related to each other in parent-child relationships where the children can be said to elaborate the detail of a parent feature. Feature model representations are often some combination of text-based, logic-based, or set-algebraic based.

Feature models are not the only modelling tool deployed in software product line development, and they are often used in combination with others e.g. use-cases, scenarios, orthogonal variability models [19]. However they have proved to be an enduring concept because they are straightforward conceptually and visually to model commonality and variability, to add additional information to each feature, o view and navigate between different levels of the forest, and to use the model to derive the features of a new product that satisfy the constraints in the feature model.

A. Customer Perspective

From a customer perspective features can be separated into different categories to help with the positioning of a product in its market segment. Kano [20] identified five categories of customer features that have different effects on customer satisfaction. "Delighting" features are value-adding, and those that have a very positive effect on customer satisfaction; "One-dimensional" features are those that customers expect to have but extra effort to improve the quality beyond industry-standard level can increase satisfaction proportionally; "Must-be" features are those that customers expect to be working at industry-standard level but extra effort to improve their quality will have little effect on satisfaction though their absence will cause dissatisfaction; "Indifferent" features are those whose presence or absence has no effect on customer satisfaction; "Reverse" features are those whose presence causes dissatisfaction. In this paper we have considered "Must-be"

and "One-Dimensional" in the same category and called them *Qualifiers* (essential but not differentiating), renamed "Delighting" as *Differentiators* (essential but differentiating), "Indifferent" as "*Don't Need – Don't Care*" and "Reverse" as "*Don't Need – Don't Want*".

A *Qualifier* is a feature that needs to be fulfilled for a product to compete in a single market segment. Often, omitting a single qualifier feature affects whether a product will successfully compete in its market segment. A product that satisfies only qualifier features is a market-entry, base product or commodity in which price becomes the sole basis for its competitiveness in its market segment (which from a development perspective relies on keeping costs low through product development efficiencies). Therefore, whilst a product needs to have the right set of qualifier features, this set is not sufficient for uniquely positioning the product in a selected market segment. For example a Qualifier feature of a mobile phone is the ability to make a telephone call.

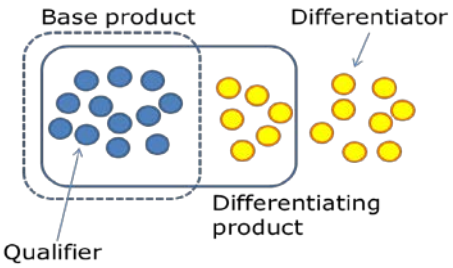

Fig. 1. Qualifier and Differentiator Features

A *Differentiator* is a feature that helps to make a product distinctive from other products in its market segment. Often this is a functional feature e.g. a TV feature in a mobile phone. Sometimes it is property of a feature e.g. its cost, quality, brand name. For example luxury brand mobile phones often have fewer functional features than top of the range mobile phones but are differentiated by packaging and casing.

A *Don't Need – Don't Care* feature arises when suppliers find it more cost efficient to provide customers with features that they do not need because they are so inter-dependent with other Qualifiers or Differentiators that they do need. If these features do not reduce some preferred aspect of quality below a desired threshold e.g. security, performance, customers are often not concerned if these features are included.

A *Don't Need – Don't Want* feature arises when suppliers provide a *Don't Need* feature that either does reduce a preferred aspect of quality below a desired threshold or there is a perception that it will do so, to the extent it leads to unwanted customer dissatisfaction.

Fig. 1 shows a set of product line features in which dark circles represent Qualifiers, light ones Differentiators. The dotted line scope shows a base product containing only Qualifier features. The solid line scope shows a differentiating product containing Qualifier and Differentiator features.

B. Supplier Perspective

From a Supplier's perspective, the set of products in a product line is often captured in a *feature model* that contains all the features of all products and in which the features are related to each other in parent-child relationships. The exact structure and content of feature models varies and often when models become complex they are partitioned e.g. to allow the connection of various layers of feature refinements, to distinguish external variability (visible to customers) from internal variability (hidden from customers) [15].

Each feature has a variability property value: *common*, *shared*, or *unique*. Common features are those that are common to all of the products in the current scope of the product line. *Shared* features appear in some but not all products. *Unique* features are in exactly one product in the product line. After time, some features can become no longer required or no longer supplied. It can be useful to denote these features as *Obsolete* for the purposes of keeping a trace history. The value of a feature model lies in the cleanliness and efficiency with which it can be used to derive the features of a new product that satisfy the feature model constraints.

TABLE I. FEATURE RELEVANCE: CUSTOMER & SUPPLIER PERSPECTIVES

Customer Perspective	**Supplier Perspective**
Qualifier	Common
Differentiator	Shared if differentiator is shared with other products in market segment, else Unique.
Don't Need, Don't Care	Common, Shared or Unique
Don't Need, Don't Want	Obsolete

A significant challenge for a supplier is to determine the appropriate property value for each feature as the product line evolves so that each product satisfies a set of customer needs but its development is aligned with the supplier's business and engineering strategy, and can be adapted efficiently without significantly effecting revenue and profit streams. Table I shows how Customer and Supplier Perspectives can be aligned with Qualifiers provided as Common features, and Differentiators as Shared or Unique features. Alignment can be difficult because of feature interconnectedness.

C. Feature evolution

As market segments and product needs change, there is an evolving tension between what a market wants (customer-led changes) and the efficiency and effectiveness with which the supplier can respond (supplier-led changes). Some changes are gradual and can be predicted, others are disruptive and cannot. Poor change management can affect performance, brand identity and reputation which in turn affects market brand positioning, pricing models, and hence sales and profits.

Customer-led changes evolve continually for different reasons e.g. new owners or executives change strategy, new legislation, competition, economic buying power, attitudinal changes in society, different cultures. For example, consumer increases in economic buying power combined with an emotional desire to take moving pictures, led to a demand by mobile phone customers (retailers) for mobile phone suppliers (manufacturers) to supply video-cameras as a standard feature in mid-range mobile phones, after a long period in which they were available only in high end phones. Retailers were no longer willing to pay a premium to manufacturers for the

video camera feature. In effect the video-camera feature moved from being a Differentiator to being a Qualifier.

Supplier-led changes also evolve for different reasons e.g. new owners or executives change strategy, new legislation, performance improvements are made to the assets in the product line architecture, new engineering tools and technologies emerge that engineering staff choose to deploy, product managers make changes because their performance pay is weighted towards reuse, technical staff expertise changes over a long period of time as people change jobs and different ideas are introduced.

Common features tend to remain unchanged and hence stable during product line evolution though their implementation may change, for example because of new technology. However the total number of common features often tends to increase over time because many product line organizations tend to add new features to a base product. Shared or unique features can be more volatile during product line evolution because their role as a distinguishing characteristic for the product in the market segment can change. For example at the same as demand for video-cameras in mid-range mobile phones increased, the cameras themselves were being reduced in physical size whilst offering the same quality and hence costing less and so it became commercially viable for mobile phone manufacturers to include them even in low-end mobile phone markets which were less demand-driven for that feature. In effect the video-camera feature moved from being a Shared feature in high-end phones to a Common feature.

Another factor to consider as a market evolves is whether a feature is *domain dependent* or *domain independent* i.e. whether a feature continues to be developed solely or principally for the product line of which it is a part (dependent) or whether it is also offered as a feature of another product line in a different market sector and its development is carried out by a third party organization (independent). In a mobile phone, an example of a domain dependent feature is the ability to make a telephone call, whereas an example of a domain independent feature is a video-camera, the development of which originated and continues to be led by companies in the photography industry.

As a market evolves, some features move from being domain dependent to domain independent. For example, software applications ("Apps") that come bundled with mobile phones were originally domain dependent and developed in-house by mobile phone manufacturers. Nowadays, a thriving "app" market has emerged in which a greater variety of better and cheaper apps have been developed by many third-parties, making them domain independent.

IV. BUSINESS STRATEGIES

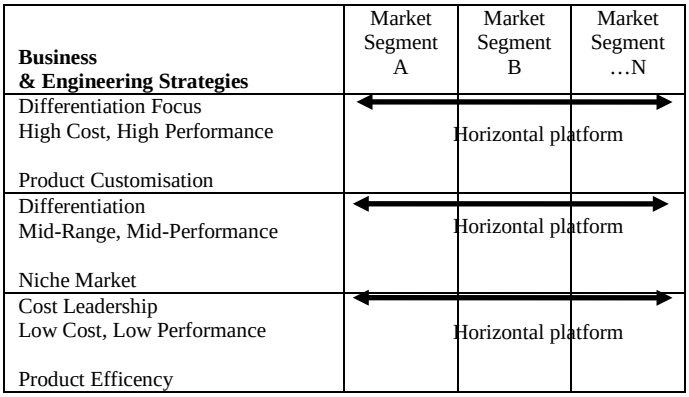

Fig. 2. Software Platform Development Strategies

The decision about which feature development option (make, buy, reuse) to deploy must be informed by the product line supplier's overarching business strategy. In Fig. 2 we have adapted the platform-market grid from [21]. The vertical axis shows three different generalized business strategies. The horizontal axis shows different market segments. We have assumed for the purpose of this paper that the platform development strategy for each choice of business strategy is to leverage systems and subsystems across market segments.

The three business strategy generalizations may not be a perfect fit to any one company but they can be helpful for analysis, understanding and evaluation of an organization's current status, future plans and trajectory.

With a Cost Leadership / Low Cost / Low Performance business strategy an organization targets a market segment and competes on price by minimizing costs. The corresponding engineering strategy is Product Efficiency characterized by making the product effective and efficient within tight budget constraints through developing a product platform in which Qualifiers and Common Features are prioritized over Differentiators and Shared/Unique features, constraining the loci for changes by reducing product variations, designing a common reference architecture, developing components that are used in each product with few variations, improving process efficiencies, getting access to a large source of low cost resources, and enhancing vertical integration.

TABLE II. FEATURE DEVELOPMENT DECISION FRAMEWORK

	Qualifier (Common - Domain Independent)	**Qualifier (Common - Domain Dependent)**	**Differentiator (Shared/Unique Domain Independent)**	**Differentiator (Shared/Unique – Domain Dependent)**
Product Efficiency	Buy	Buy	Buy	Make
Niche Product	Buy	Reuse	Buy/Reuse	Make
Product Customization	Buy/Reuse	Reuse/Make	Reuse	Make

With a Differentiation / Mid-Range / Mid-Performance business strategy an organization targets a set of market segments with a product that has a unique combination of Qualifiers and Differentiators and price that is valued by customers. The corresponding engineering strategy is Niche Product characterized by a product platform with an architecture that is designed for high numbers of Qualifiers and Differentiators i.e. high variation management and asset-based reuse, but with a focus on quality and innovation for the feature combinations on offer. Each product is formed by tailoring selected assets from a base of common assets using pre-planned variation mechanisms, adding any new components that may be necessary, and assembling the collection according to the rules of a common, product-line-wide architecture.

With a Differentiation / High Cost / High Performance business strategy an organization targets a market segment but provides specific products to suit the needs of individual customers generating high customer loyalty allowing them to pass on higher costs and charge higher prices. The corresponding engineering strategy is Product Customization characterized by a product platform to support a compositional approach of increasing numbers of Differentiators, in which variability management becomes de-centralized and the product line architecture only guarantees that the components can be connected together. Over time each product assumes its own maintenance trajectory separate from other products, there is little or no product line or reuse, and the technical strategy is effectively single-system development with some reuse.

V. FEATURE DEVELOPMENT FRAMEWORK

Table II shows a feature development framework to provide advice to suppliers for different overarching business and engineering strategies. The framework assumes that reliable third parties exist from whom features can be bought or reused at an acceptable level of quality and with whom satisfactory contractual arrangements can be established.

A. Qualifiers – Domain Independent

For Qualifier features that are domain independent, *buy*ing often becomes the preferred choice regardless of strategy to enable greater economies of scale to be achieved. Qualifier features represent a sound basis for reuse and hence to form part of a product line platform. However, from a commercial perspective the value of Qualifier features is often low. Few customers are willing to pay a premium for these features being met. The implication is that they should be realized at the lowest cost possible. The competitor that can achieve the greatest economies of scale in creating these features is likely to have competitive advantage. However for a product customization there may be instances when reuse is required.

B. Qualifiers – Domain Dependent

For Qualifier features that are domain dependent the decision on the development approach is more difficult. In principle the widespread inclusiveness of a Qualifier feature does not make it an attractive proposition to create a tailor-made new solution and deploying a domain dependent technology may reduce the future ability to acquire technologies. For all strategies the preferred option is to buy, and when this is not possible then reuse is most likely, although occasionally, it may be that make is the cheapest option. A preferred compromise arises when the product line organization has previously created other legacy systems, products or technologies which can be reused, perhaps after refactoring. Sometimes, if market circumstances are suited, suppliers may collaborate to implement a set of Qualifiers that they recognize must be in every product regardless of the supplier. For example in the automotive industry, many automobile manufacturers, suppliers and tool developers collaborated to develop a standardized automotive architecture to assist with developing vehicular software [22]. This enabled partners to benefit from reduced costs for the Qualifiers against other global suppliers not in the consortium, and to focus on competing on Differentiators i.e. specific software applications in each vehicle. The less the sharing across competitor products the less likely this approach will succeed.

C. Differentiators – Domain Independent

For Differentiator features that are domain independent, *buy* or *reuse* is the preferred strategy and the choice between the two is often determined by the drive to keep costs low versus the complexity and seamlessness of integration which require some adaptation. But the expertise may require adaptation depending on the existing configuration and properties of the product line it is being integrated with.

D. Differentiators – Domain Dependent

For Differentiator features that are domain dependent then *make* is the preferred strategy. These features will often be a source of competitive advantage. Even if the business strategy is cost leadership, for features that provide a product with unique characteristics then making the feature product specific can be more efficient and effective than trying to reuse and adapt similar realizations. For example most mobile phone manufacturers are exploring new packaging and presentations of existing functionality e.g. curved screens, taking advantage of elasticity properties in new materials.

TABLE III. FREQUENCY CONVERTER FEATURES CATEGORIZATION

Domain	Qualifier (Common - Domain Independent)	Qualifier (Common - Domain Dependent	Differentiator (Shared/Unique Domain Independent	Differentiator (Shared/Unique – Domain Dependent
General frequency converters	Operating system (OS) Generic middleware (MW) Typical field busses (FB)	Basic motor control (MC) Basic application behavior (AP)	Programmability (P)	Advanced motor control (AMC) Advanced application behavior (AAP)

VI. Case Study: A Frequency Converter Product Line

Danfoss Power Electronics A/S (DPE) is a division of Danfoss Corporation, a Danish company with more than 24000 employees around the world. DPE manufactures a wide range of electrical and electronic products including industrial frequency converters. Frequency converters are used to change the frequency and magnitude of the constant grid voltage to a variable load voltage and hence to optimize the speed of electric motors used to control many industrial processes. If there is no frequency converter, then the electric motor runs either at full speed or is not running. Frequency converters allow savings of up to 60% of the energy consumed in an electric motor, a significant statistic when considered against the global volume of electric motors used in industrial processes. Fig. 3 shows a typical industrial automation application that has many mechanical parts, and is driven by an electric motor using some kind of transmission.

Frequency converters have a long lifespan e.g. 15 years, and the frequency converter platform tends to change significantly during this time (in terms of functionality, hardware, software infrastructure). A key challenge is to take advantage of advances in technology within the frequency converter without modifying its external behavior within the customer system it is deployed. For example some customers want a new frequency converter with some new features but also want it to have exactly the same behavior including timing characteristics as the old frequency converter. Other customers require the same features but with faster response times to run new applications. Additional complexity arises when customers combine frequency converters from different generations without updating the software of the old frequency converters.

We used the framework set out in Table II to evaluate the construction of groups of product features of three different frequency converter product lines. Product Line A means a representative product of a product line constructed within the context of a Product Efficiency business strategy. Product Line B means a representative product of a product line constructed within the context of a Niche Product business strategy. Product Line C means a representative product of a product line constructed within the context of a Product Customization business strategy. To simplify discussion we focus on the differences in the construction decisions for groups of product features that occur in each of the product lines. In practice our initial intention was to make the analysis only on the feature space, to serve as an input to the architecture design phase. However we realized that we needed to iteratively evolve both the feature specification and the architectural design. The entire process took us about 6 months to complete. Table III shows examples of frequency converter feature groups for each of the four different feature variability types.

A. *Qualifiers (Common - Domain-Independent):*

Each frequency converter requires operating system services, to get access to the basic memory, processing, and communications services of the underlying hardware. Often, an abstracted communication system is provided in terms of basic middleware services that may hide the deployment of software components for more distributed system architectures. In addition, the support for basic field buses (e.g. those that adhere to PROFINET the open standard for industrial Ethernet [23]) and analogue/digital I/O terminals are common domain independent features that nearly all frequency converters are expected to support.

B. *Qualifiers (Common - Domain Dependent)*

The main behavior of a frequency converter is that it changes the speed of an electronic motor. For this, it needs motor control algorithms that are specific to the motor type (e.g. permanent magnet motor or induction motor). The basic ability to control one motor type is a qualifier that is common to all frequency converters, but clearly domain dependent. Similarly, basic application control e.g. controlling a fan by giving a desired RPM rating is a qualifier but domain dependent.

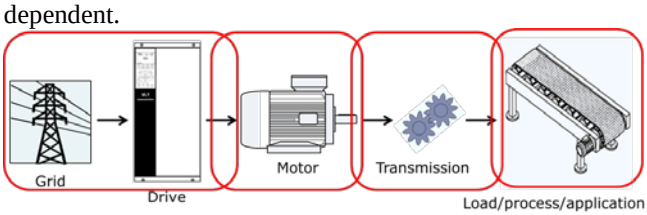

Fig. 3. A typical industrial application with a frequency converter

C. *Differentiators (Shared/Unique Domain Independent)*

A Programmable Logic Controller (PLC) allows the deployment of standardized simple programming languages to create, modify or extend the application behavior of the frequency converter. Drive manufacturers that support PLC programmability can increase the value of their frequency converter for customers. However, PLCs are common across many industrial automation environments thus making them domain independent differentiators.

D. *Differentiators (Shared/Unique Domain Dependent)*

There are also domain-specific differentiators that are an area of continuous development to guarantee sufficient differentiation and value-add compared with the competition. Advanced motor control and application control features belong to this category. For example in some applications it can be differentiating to offer motor controls that have a fly-start i.e. an ability to start controlling an already moving motor, or in vertical lift applications it can be differentiating to offer functions such as load compensation or reduction of swinging of the load.

VII. Results

Table IV shows the result after our analysis. The bold text in each cell represents the original recommendation of the framework. The other values represent what was actually done for each of the product lines. Broadly, the framework's recommendation was consistent with actual practice. The shaded boxes show where there were significant differences.

A. Qualifiers (Common - Domain-Independent)

Product Lines A and B used open source derived software for their middleware and operating systems. Product Line C did not have an operating system or basic middleware behavior at all. It was to be deployed in a highly resource-constrained environment so it had chosen a bare metal implementation without abstracting the hardware at all and implemented directly only the required basic services to reduce both memory and processing power requirements to the absolute minimum. So while in general these basic functions are not differentiating and the framework holds, in this particular case the need to aggressively reduce memory and use a smaller microprocessor (to reduce the bill of materials and hence cost) led to a customized development. We say more on this in Section VIII.

The basic field buses were initially created for Product Line B. Typically the basic fieldbus stack was purchased but the organization was responsible for integrating this field bus into the frequency converter. As the fieldbus implementations existed already in-house Product Line A reused it directly. Product Line C did not have a basic fieldbus interface only so customer specific realization was required.

B. Qualifiers (Common - Domain Dependent)

When the development work for Product Line B started, there were no commercially available basic motor control implementations. Since advanced motor control features are a significant differentiating factor, creating a structure for the basic motor control that can be easily extended for advanced functions was important and therefore each of the product lines had their own basic motor control features.

The implementation for Product Line B is the most extensive as it also has the most advanced motor control features. When implementing the basic motor control for Product Lines A and C the developers had detailed knowledge of the motor control of Product Line B. This made the motor control structures of Product Lines A and C more optimized subsets of the motor controls of Product Line B.

In hindsight, some refactoring of the implementation of Product Line B in both motor controls of Product Lines A and C could have been made using a common asset. However at the time *independent execution was favored over reuse*.

Similarly, as the initial application focus was different for the different product lines they produced completely independent implementations of the basic application behavior. Also for Product Lines B and C the ability to easily extend the application behavior with advanced application functions drove the decision of independent implementation.

C. Differentiators (Shared/Unique Domain Independent)

Only Product Line B has a programmable environment that allows the customers to create their own programs that will then run inside the frequency converter. This environment and the corresponding programming tools have been acquired from an external company.

D. Differentiators (Shared/Unique Domain Dependent)

For Product Line A there are no advanced motor or application control features as the product line is aimed at the low cost market and differentiating domain specific features are not needed here. Product Lines B and C have made both their advanced motor control and application control functionality independently as also the basic functionality for both was of independent development.

VIII. Discussion

The framework presented in Table II is a useful tool to help to take stock, expose and re-evaluate the role of reuse activity as a contributory factor to the underlying supplier business model. We recognize it provides only general level guidance on approaches to the realization of features and that the final decision to make, buy or reuse must factor in other issues as set out in [14].

TABLE IV: Frequency Converter Features - After Analysis

Product line	Qualifier (Common - Domain Independent)	Qualifier (Common - Domain Dependent	Differentiator (Shared/Unique Domain Independent	Differentiator (Shared/Unique – Domain Dependent
A - Product Efficiency	**Buy** (OS) Buy – Open source (MW) Open source (FB) Buy (Reuse)	**Buy** (MC) Reuse/Make (AC) Make	**Buy** (P) N/A	**Make** (AMC) N/A (AAC) N/A
B - Niche Product	**Buy** (OS) Buy – Open source (MW) Buy - Open source (FB) Buy	**Reuse** (MC) Make (AC) Make	**Buy/Reuse** (P) Buy	**Make** (AMC) Make (AAC) Make
C- Product Customization	**Buy/Reuse** (OS) Make – bare metal (MW) N/A (FB) N/A (only I/O)	**Reuse/Make** (MC) Reuse/Make (AC) Make	**Reuse** (P) N/A	**Make** (AMC) Make (AAC) Make

OS – Operating System MW - Middleware FB – Field Bus
MC – Motor Control AC – Application Control P – Programmable Environment
AMC – Advanced Motor Control; AAP - Advanced Application Control

In general the advice to buy in a solution when a Qualifier is Domain Independent is sound, this allows internal developer focus to be maintained on creating differentiating features. However if differentiating functionality relies significantly on the structure of how the basic qualifying functionality is implemented it tends to drive a decision to keep creating also qualifier functionality. The main reason is to guarantee that the differentiating functionality and the interface between the qualifier and differentiating features can be created quickly and with good quality. Having full control of assets also assists transforming previously differentiating functionality to qualifiers as the markets change. We noticed that as more and more application functions are added to frequency converters, many of them become Qualifiers.

Having existing assets within the company tends to increase the likelihood of reuse, more so than buying in assets externally. However, one must also consider, besides the initial cost, the amount of future changes and maintenance investment i.e. the degree to which a feature is stable or volatile. In principle the more volatile a feature the more likely the decision is concerned with the risks and costs of managing several changes. Passing these risks on to a third party provider can be an attractive option but much depends on the price premium that the third party provider will place on managing this risk compared to managing it in-house.

Successful product lines tend to evolve over long period of time. We observed that over time standardization and commercial offerings tend to emerge that often cover non-differentiating functionality and drive down the cost and by doing so effectively transform previously differentiating functionality into qualifier. New choices emerge during the evolution of the product line that were not available initially. Organizations are typically very reluctant to abandon software that has been used to create value even if it would be highly beneficial. However external efforts to standardize functionality may force a company to adopt a commercial offering instead of its own in-house solution. For example the standardization of application functionality from the OpenPLC consortium may eventually force all the product lines in our case study to change their qualifier functionality to match the interface of the OpenPLC function blocks [24]. A variation on this theme is to split original features into domain independent and domain specific parts based on feedback by software and hardware architects, allowing new opportunities to outsource some of the development work. The lesson here is to continue to explore new ways to realize existing features in architectural design and implementation.

Organizational structures can also significantly affect development decisions. If product lines are geographically distributed and independently managed, product line managers may often make product line development decisions principally from the perspective of their own local interests without considering the impact on other market segments. In this case study, the decision to create a bare metal implementation for Product Line C rather than reusing the operating system from product line B would probably not have happened if greater global, cross-platform governance had been in place.

Changes in hardware capabilities change tend to drive reuse or buy options. Historically, embedded systems have been using highly dedicated solutions. However, frequent increases in processing power and memory are starting to enable the use of more standard solutions originating from non-embedded domains.

Earlier we remarked that the evidence from a range of empirical studies [2] had noted that *make* was more likely to occur the greater the asset specificity. We do not claim to have conducted an empirical study but our experience from the case study seems to support that assessment.

IX. THREATS TO VALIDITY

We recognize that in this qualitative research using three case studies that there are some threats to its validity.

Objectivity: The formation of this feature development framework was based on the authors' experiences of the telecommunications domain over multiple projects and product lines. The authors were not involved in any of the case studies described and hence not involved in any of the make/buy/reuse decisions made in each one.

Generalizability: We selected three case studies that we viewed as representative of each of the three business strategy dimensions. Given that DPE has constructed a wide range of industrial frequency converters for different clients in different settings we recognize that a sample size of three is small but the complexity of each case constrained what we were able to examine in the limited time available. We also recognize that what we selected as *representative* case studies might be viewed differently by other researchers who might make other choices.

Credibility: Our understanding about the decisions that were made and their rationale was formed by documents and interviews with some of the practitioners who worked on each case study. However some of the decisions on these long-lasting product lines were taken 10 years earlier by people that are no longer employed by DFE, and we needed to rely on other colleagues' interpretation about why such decisions were taken which may be different from the actual original reasons. In addition, we did not share the framework with the interviewed practitioners, and so the categorization of the make/buy/reuse decisions was done by the researchers and poses a threat of incorrect categorization for each decision. We also acknowledge that when we applied the framework to each case study we did so at a particular point in time. We have not evaluated the application of the framework on the same case study over a sustained period of time.

X. CONCLUSIONS

Whilst a corporate engineering commitment to reuse is at the core of product line development, reuse is simply an enabler for profitability of the product line organization and it is important to find the right balance so that reuse is not undertaken for its own sake. In a fast-moving product space, where first mover product feature advantage can make a

significant difference to market share, the development efficiency of the unique differentiation characteristics is likely to take precedence over development for reuse. This typically means some form of agile product specific development on top of the common platform.

Traditional product line scoping has focused on variability of the product line features and the assets that realize those features. We believe that splitting features into qualifying and differentiating features for each market segment assists in making the right decisions in terms of make, buy or reuse decisions for real industrial product lines. In this paper, we e have presented a framework to model possible choices related to making, buying and reuse features based on the business strategy of the product line. The framework was reassuringly consistent in its recommendations with actual practice for 9 out of 12 development decisions. However just as important is that the framework provided a vehicle for helping stakeholders see development choices within the broader evolving context of the business strategy from the supplier's perspective and changing needs from customers' perspectives, all of which in turn led to better decisions being made by the stakeholders involved. We acknowledge that in some instances the recommendation proposed by the framework did not align with actual practice, and we are currently testing the framework against other product lines to understand the extent we may need to modify it.

REFERENCES

[1] R. H. Coase. "The Nature of the Firm", in The Firm, the Market and the Law. Chicago: University of Chicago Press, 1937.

[2] P. Klein, The Make-or-Buy Decision: Lessons from Empirical Studies in C. M´enard and M. Shirley (eds.), Handbook of New Institutional Economics, Springer, 2005, pp435–464.

[3] O. Williamson, "Strategizing, Economizing, and Economic Organization", Strategic Management Journal, vol 23, 1991, pp. 75–94.

[4] M. Bidwell, "Problems Deciding: How the Structure of Make-or-Buy Decisions Leads to Transaction Misalignment Organization Science", Articles in Advance, 2009 INFORMS, pp. 1–18.

[5] G. T. Fechner, H. E. Adler, D.H. Howes, and E.G.Boring, Elements of Psychophysics. Holt, New York, 1966

[6] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product- Line Approach. Addison-Wesley, 2000.

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

[8] I. John, J. Knodel, T. Lehner, and D. Muthig, "A Practical Guide to Product Line Scoping", Proceedings of the 10th International Conference on Software Product Lines, 2006, pp3-12.

[9] J-M. DeBaud, and K. Schmid, "A Systematic Approach to Derive the Scope of Software Product Lines", Proceedings of the 21st International Conference on Software Engineering, 1999, pp34-43.

[10] K. Schmid, and M. Schank, "PuLSE-BEAT -- A Decision Support Tool for Scoping Product Lines", Software Architectures for Application Product Families, Lecture Notes in Computer Science 1951, Proceeedings of the International Workshop IW-SAPF-3, ed F van der Linden, Springer, 2000, pp65-75.

[11] K. Schmid, S. Thiel, J. Bosch, S. Johnsson, M. Jaring, B. Thome, and S. Trosch, Scoping, Eureka Σ! 2023 programme, Deliverable 1.2.4, ITEA 99005, ESAPS, 2001.

[12] J. Lee, J, S. Kang, and D. Lee, "A Comparison of Software Product Line Scoping Approaches", International Journal of Software Engineering and Knowledge Engineering, Vol. 20 No.5, 2010, pp637-663.

[13] S. Comella-Dorda, J. Dean, G. Lewis, E. Morris, P. Oberndorf, and E. Harper, A Process for COTS Software Product Evaluation, Carnegie Mellon University Software Engineering Institute, Technical Report SEI-2003-TR-017, July 2004.

[14] http://www.sei.cmu.edu/productlines/frame_report/MBMC.htm (last accessed 1 January 2015).

[15] K. Pohl, G. Böckle, and F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005

[16] I Schaefer, L. Bettini, L.V. Bono, F. Damiani, and N. Tanzarella, "Delta-oriented Programming of Software Product Lines", Proceedings of the 14th International Conference on Software Product Lines, Sept 2010, pp 77-91.

[17] K. C. Kang, S G Cohen, J. A. Haess, W. E. Novak, and A S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute Technical Report, Carnegie Mellon University Software Engineering Institute, Technical Report, CMU/SEI-90-TR-021, Nov 1990.

[18] P. Clements, Product Line Engineering Comes to the Industrial Mainstream http://www.biglever.com/extras/PLE_Industrial_Mainstream_Whitepaper.pdf (last accessed 30 Sept 2015).

[19] S. Buhne, K. Lauenroth, and K Pohl, "Modelling Requirements Variability across Product Lines", Proceedings of the 13th IEEE International Conference on Requirements Engineering, 2005, pp. 41-52.

[20] N Kano et al., "Attractive Quality and Must Be Quality," The Journal of Japanese Society for Quality Control, 1984, VOL?? pp. 39-48.

[21] The Power of Product Platforms, M Meyer and A. P. Lehnerd, Simon & Schuster 1997.

[22] www.autosar.org (last accessed 1 July 2015).

[23] http://w3.siemens.com/mcms/automation/en/industrial.-communications/profinet/pages/default.aspx (last accessed 1 July 2015).

http://www.plcopen.org/index.html (last accessed 5 July 2015).