Evolution of a Bluetooth Test Application Product Line: A Case Study

Narayan Ramasubbu, Rajesh Krishna Balan
Singapore Management University
80 Stamford Road, Singapore 178902
[nramasub, rajesh] @ smu.edu.sg

ABSTRACT

In this paper, we study the decision making process involved in the five year lifecycle of a Bluetooth software product produced by a large, multi-national test and measurement firm. In this environment, customer change requests either have to be added as a standard feature in the product, or developed as a special customized version of the product. We first discuss the influential factors, such as evolving standards, market share, installed-base, and complexity, which collectively determined how the firm responded to product change requests. We then develop a predictive decision model to test the collective impact of these factors on determining whether to standardize or customize a customer's change request. Finally, we develop and test a customization cost estimation model, for use by software product teams, which specifically accounts for factors unique to the customization stage of a product lifecycle.

Categories and Subject Descriptors

D.2.9 [**Management**]: Life cycle, programming teams, software process models, software quality assurance

General Terms

Economics, Management

Keywords

Software process, software engineering economics, complexity, product development, product life cycle, software evolution

1. INTRODUCTION

A packaged software product refers to a software application that is developed for mass distribution, and not specifically designed to fulfill the unique needs of an individual customer. Packaged software manifests itself in many domains including Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), and Supply Chain Management (SCM) systems. As such, packaged software is a big business – for example, the worldwide market for packaged systems software products alone is estimated to be more than $10 billion [10]. In addition to the stages of a typical software development lifecycle (requirements analysis – design – build – test – maintain), a packaged software product lifecycle includes the standardization and customization phases, which have not yet drawn much attention from the software engineering research community.

FSE-18, November 7–11, 2010, Santa Fe, New Mexico, USA.

Customization refers to the process of modifying the generic product functionalities found in a packaged software application to fulfill the unique needs of a customer. Over time, the packaged software product vendor might release these individual, customer-specific modifications to all its installed-base by 'standardizing' the feature in its future release. When a customer raises a change request, a packaged software product vendor, thus, has to decide whether to fulfill the customer request by offering a customized solution to the individual customer, or to offer a standardized solution by releasing a new feature in a future version of the product. This standardization-customization decision becomes significantly complex in the presence of heterogeneous customer needs, compatibility constraints, and evolving standards.

In this paper, we analyze the standardization-customization decisions made by a vendor over the lifecycle of its Bluetooth test and measurement software product. The lifecycle of this product was marked by several uncertainties including the rapidly evolving Bluetooth standards specifications and a diverse customer base. Thus, it provides a rich empirical context to understand the factors that influence the standardization-customization decisions of a vendor. Further, by analyzing how the firm responded to change requests from customers, in the face of evolving standard specifications, we are able to draw lessons and recommendations for both product management and managing complexity in software product development.

The three main contributions of this paper are:

1. We present a detailed study showing how a technology company organized its processes to deal with constantly changing specifications originating from standards evolution and diverse customer requirements.
2. We develop and test a predictive decision model accounting for the factors that influence the standardization-customization decision of a software product vendor, and thus advance a step towards a generalizable theory of software product line evolution.
3. We develop and test a customization cost estimation model that fairly accurately predicts the customization costs of packaged software product, contributing to the advancement of our understanding of the economic aspects of modeling variability in product line evolution.

2. RESEARCH SITE

2.1. Company and Product Overview

Our research site, Measuretronics (not a real name due to non-disclosure agreements), operates in about twenty countries. This large company has over $1.5 billion in recent annual sales and

holds about seven hundred active technology patents. Measuretronics invests about a third of its revenues in research and development across five businesses (oscilloscopes, logic analyzers, video test products, telecommunications equipment, and optical sensor products). The firm has about 400 customers in various industries including electronic standards compliance testers, semiconductor chip manufacturers, communication equipment manufacturers, and telecommunication application developers. The product we study in this research was developed by Measuretronics, and is called the "Bluetooth protocol analyzer". Bluetooth is a short range wireless communication standard that allows connectivity between electronic equipment [3]. The Bluetooth protocol analyzer is a test and measurement equipment used during the production of Bluetooth consumer products. Typical customers of the Bluetooth protocol analyzer are not consumer end users, but the firms producing Bluetooth-enabled consumer goods. Firms producing Bluetooth devices need to test the interoperability of their devices with other Bluetooth devices that may be used in the consumer's personal area network. This test procedure involves analyzing the radio frequencies transmitted and received between the networked devices. The Bluetooth protocol analyzer facilitates this procedure by intercepting the traffic between Bluetooth devices and enables detailed engineering and statistical analysis of the intercepted communication data.

The Bluetooth protocol analyzer product consists of a hardware component and a software component. The hardware component comprises of an air-sniffing probe to intercept wireless communication traffic between Bluetooth devices, a cable-sniffing interface to intercept wired communication traffic, and serial and parallel data port interfaces for connecting to a personal computer. The software component involves a firmware operating system that is stored in the memory of the hardware component, and a packaged software product that needs to be installed on a client personal computer. The physical hardware and the firmware together intercept and capture data from traffic between two Bluetooth devices. This data is passed to the client software package for further analysis. The software package is used to define specifications, represent results in a graphical manner, and prepare testing reports.

The development of this product was started in 1999 and continued until 2004 when the product was eventually shelved by Measuretronics. It should be noted that the Bluetooth protocol specification, was constantly changing during this time period. For example, the Bluetooth Special Interest Group (SIG) was only formed in 1998. The Bluetooth 1.0 specification was released in 1999, the 1.2 specification was adopted in 2003, and the 2.0 specification was adopted in 2004. A full history of the Bluetooth standard and feature changes is available elsewhere [2].

2.2. Product Development Process

The base product development process followed at Measuretronics involved three major steps. The first step involved collection and consolidation of customer requirements. The firm uses an online system called "customer support network" to log all customer requests and feedback. When a customer enters the details of a request in the system, a unique ticket number is created. These feature request tickets are consolidated on a weekly basis. The second step in the product development process involves a thorough analysis of the feature request tickets. This involves removing duplicate tickets, grouping tickets from different customers that have similar content, and validating the content of each customer ticket. If the requested feature is already present in any of the released versions, a response is sent to the customer and the feature request ticket is closed.

In the last step, the product management team works in tandem with the development team to a) analyze the technical feasibility of the change request, b) check if the requested feature is shipped by a competitor, and c) generate an appropriate customer survey for the requested feature. The customer survey is designed to obtain installed-base feedback on the proposed feature, and is emailed to the existing customers of the product. Customers are asked to give feedback on whether they find the proposed feature (or change) useful. To discourage casual feedback or non-response, customers are also informed of the impact of the proposed change in terms of code modifications and future service pack installations. After consolidation of installed base opinion, a final decision is made on the fulfillment of the request.

2.3. Data Collection

As a first step, we established the research protocols for the study, which included details on participant selection, roles and responsibilities among participants, interview structure, archival data collection rules, structuring learning opportunities, and implementing recommendations. We adopted a participatory action research methodology. Participatory action research enables researchers to co-investigate or involve the communities whose practices they study in their research activities. These activities include data collection, record keeping, model building, solutions development and implementation. This research methodology allows researchers to obtain a deep understanding of the problem context, with a view to make improvements in the practices, even when they do not, a priori, have well defined solutions or theories.

We formed a three-member core team consisting of one of the authors and one manager each from the program management and engineering division of our research site. The program manager was the senior most personnel involved in the development of the Bluetooth protocol analyzer product, and had a global view of product development process. The engineering manager worked independent of the program manager (i.e., did not have direct reporting responsibility), and had in-depth knowledge about the technicalities of the product. The core team was responsible for the overall conduct of this study, and recruited additional participants at different stages of the study. Learning opportunities were structured using a three-phase cycle consisting of diagnose-take action-evaluate stages.

During the course of the study, the core team conducted ten structured interview sessions with project leaders, engineers and product managers of the Bluetooth protocol analyzer team. The first four of our interviews happened during the active product release cycles (one each from 1999-2003). The rest of the interviews were conducted after the active product lifecycle years (the most recent one in December 2009). Our archival data collection process was spread throughout the product release years, until early this year. We recorded all the interviews, and the archival data was organized using a version control system to help us not lose track of the activities spreading multiple years. One of reasons for the long delay between end of the product lifecycle and this case study reporting process is the mandatory requirement of a five-year gap as per the firm's non-disclosure agreement signed

with us. Further, the firm was recently acquired by another multi-national company leading to new regulations and clearance procedures. Because we facilitated good record keeping, and were able to pull-up objective data from the process database at any time, we are confident that the long gap in the reporting process does not threaten the reliability and validity of our research data. Moreover the core research team including the participants from the firm did not change throughout this project, which further facilitated good organizational memory for this study.

Each structured interview session with the project managers and software developers lasted about an hour. Before each interview session, the core research team organized the questions and collected all relevant archival data related to the discussion. Each of our archival data analysis and interview sessions had a "diagnose'' phase and a "evaluate" phase. The "diagnose" phase of the sessions were structured to understand the specific practices of the product team during the prior release cycles and discuss the experiences of the team. In the "evaluate" phase participants had to correlate their experiences with the archival data of the process database and reflect on the matches and deviations. The data collected through these sessions, along with the archival process database at the firm form the basis for this case study analysis. Thus, our collected data spans the entire life cycle of the Bluetooth analyzer product and includes the following:

- In-depth records of the functional features and specification changes of the Bluetooth protocol analyzer. These were obtained from the company's product database.
- In-depth process records detailing the product development processes and choices made. These were obtained from the company's process databases and the structured interviews.
- Interviews with product managers, program developers, customers, and Bluetooth domain experts to provide more qualitative insights on our data points. These interviews also helped us validate the quality of the process data collected.

On an average, the Bluetooth product line team at Measuretronics had a full time equivalent head count of 13 personnel (8 developers, 1 usability engineer, 2 project managers, 1 program manager, and 1 product marketing manager). The core product team worked with other divisions of the firm, including the program management office and marketing department, and occasionally borrowed more resources from other divisions.

There were 203 customer generated product requests made during the product's lifecycle. With the assistance of the product management team at our research site, we filtered out duplicate requests and ended up with 154 unique customer requests. We then analyzed the processing history of these requests and categorized them in to two divisions: (1) customer requests that were decided to be standardized into the next product release, and (2) customer requests that were processed as customization projects through contractual relationships with individual customers. There were 92 instances where the firm decided to process customer requests as standardized product features in future releases, and 62 customer requests that were handled as customization projects. We then collected cost and development personnel related data from the 62 customization projects. The product and program management departments could not furnish data for 9 of these projects. So our final data set includes data on 154 standardization-customization decisions during the product lifecycle, and 53 customization projects executed by the firm.

Our final data collection step was the analysis of the software code to collect data on complexity and individual program related metrics. An exhaustive analysis was made for 56 different versions of the source code. The metrics we collected were verified individually by two domain experts at our research site. Whenever there was a mismatch, the source code was pulled again from the repository and checked for inconsistencies. Later, these collected metrics were authenticated using an independent object oriented metrics collection tool. Any mismatch that arose from the authentication was checked by a technical expert at our research site and rectified.

3. FACTORS INFLUENCING SOFTWARE STANDARDIZATION- CUSTOMIZATION

In this section, we discuss the influential factors affecting the standardization-customization decisions at Measuretronics. We identified these factors by analyzing our case data, and by interviewing product management and engineering teams about their decision making process when presented with individual customer requests.

3.1. Evolving Standards

Being a test and measurement company, Measuretronics had to develop its test application even when the Bluetooth standard specifications were not well established and accepted. Under this uncertainty, Measuretronics product management was constantly facing customers who requested features that violated Bluetooth standards existing at that time. Releasing features which violated Bluetooth standards would jeopardize Measuretronics' official position in the Bluetooth standards making process. At the same time, some of the feature requests came from business units of influential, multinational customers who were loyal to Measuretronics for multiple years. Finally, there were other competitors in the market who boldly offered features that violated the Bluetooth standards of the time (and marketed those violations as positive features). Hence it was difficult for the Measuretronics product management team to enforce a uniform policy on features that violated Bluetooth standards. In the absence of a uniform standards-violation policy, the engineering team started playing an important role, along with the product management team, in deciding the fate of feature requests. This created additional coordination overheads in the product decision-making process.

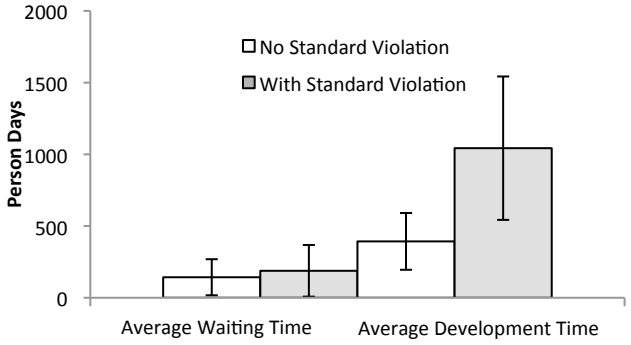

Figure 1. Effect of Standards
The error bars indicate standard deviations. Person days are calculated for a team of 10 developers.

Figure 1 shows that the customer requests that asked for features violating the existing Bluetooth standards usually took significantly longer to be fulfilled. We also observed that the heuristics employed by the product management and engineering teams for determining the features that violated standards depended on several other factors, including competition, compatibility, the nature of change involved, and complexity considerations. We discuss these factors subsequently, and model their collective impact on the standardization-customization decision in Section 4.

3.2. Competition

Our analysis of how Measuretronics reacted to competition shows interesting patterns as shown in Figure 2. The "No Competition" bars describe features for which Measuretronics had no competition while the "With Competition" bars describe features for which there was viable competition. First, we notice that more than half of the features in Measuretronics's product had no viable competition. It is interesting to note that although the average wait-time for a customer request to be implemented was about the same with or without competition, the development time for product features that had competition was significantly lower as compared to the development times of features that were unique to Measuretronics. This indicates that the Measuretronics product teams were particularly sensitive to competitive pressures, and wanted to stay ahead in the market by uniquely positioning the product. We also note that the firm kept its standard violations to a minimum in the presence of competition – again to uniquely position the product as being better standards-compliant than its competitors.

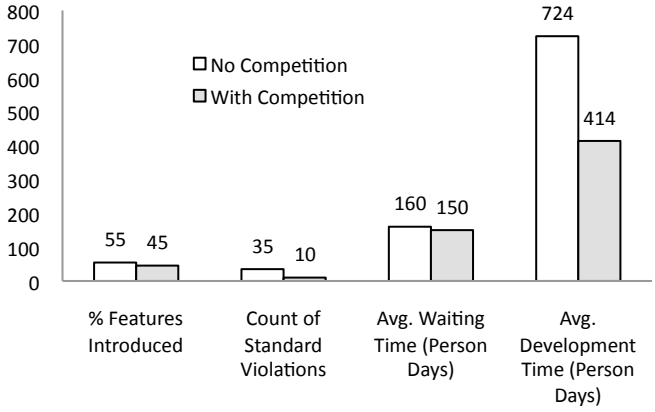

Figure 2. Effect of Competition
Note: Y-axis units vary and are stated below each pair

3.3. Compatibility

For every customer request that is received, Measuretronics also assessed whether the proposed feature would cause incompatibility with previous releases of the product, i.e., violate backward compatibility. We noticed that in the initial stages of the product evolution, Measuretronics was more diligent in providing backward compatibility. Features that violated backward compatibility were more often stripped in the standard version, and were provided through a custom release for specific customers. However, as the market share of Measuretronics's product increased rapidly, several features with severe backward compatibility problems were standardized across the board for all customers. For example, we noticed a 25% increase in backward incompatible features in the fifth release of the product as compared to the previous four releases. While part of the reason for increase in violations of backward compatibility could be because of the way Bluetooth standards evolved, it also appears that Measuretronics used its dominant market share position to standardize incompatible features in a general release.

3.4. Nature of Change

The increased participation of engineering teams in product management decisions at Measuretronics, as noted before in Section 3.1, added an interesting dynamic to the analysis of customer feature requests. Along with the marketing and profitability considerations, the engineering nature of potential product changes also became a central focus. Our observation of the product release strategy meetings revealed that the Measuretronics teams carefully classified the effect of a customer requested feature on a release in to four categories of changes: (1) Incremental changes, (2) Modularity changes – changes which affected significant portions of self contained modules of the product, (3) Architectural changes – changes which significantly affected the fundamental software architecture of the product, and (4) Radical changes – changes which significantly disrupted the entire product ecosystem including the software, hardware and development platforms.

To see if this engineering consideration of changes had any impact on fulfillment of customer requested features, we analyzed the customer waiting times and feature development times for each category of changes. The pattern of waiting and development times across the four different categories of changes is shown in Figure 3. We notice that Measuretronics gave priority attention to changes that affected modularity and architectural changes (lowest development time bars – even lower than incremental changes), indicating that there was an active effort to maintain the structural complexity of the system.

Figure 3. Nature of Change

3.5. Complexity

In our interviews with the Measuretronics teams, complexity was repeatedly mentioned as one reason why certain customer requests went unfulfilled even when the marketing reasons for fulfilling those requests were compelling. Also, we noticed that a simple complexity-based heuristic was used by the Measuretronics engineering teams to decide on fulfilling customer requests. Customer requests that were functionally complex – i.e., those demanding complex algorithms and data structures tended to be satisfied through customization. On the other hand, customer feature requests that involved high levels of structural complexity –

i.e., those involving complex cross modular interfaces tended to be fulfilled through standardized general releases. The application of this simple complexity-based heuristic to decide the standardization-customization strategy seems to stem from Measuretronics's quality management considerations. Managers rationalized that fixing errors arising from higher levels of functional complexity often required close relationship with customers to understand their full implementation set-up (the networked devices and the related software applications). Hence, product managers preferred custom solutions for customer requests that involved high levels of functional complexity. On the other hand, Measuretronics engineering team had better internal capabilities to understand the overall structure of their product than any one single customer. Hence, when a customer feature demanded structural alterations, standardizing structural changes was seen as more appropriate than a customized solution.

4. STANDARDIZATION-CUSTOMIZATION DECISION MODEL

Our interaction with the Measuretronics managers helped us discover the key factors that influenced their decisions. While we discussed these individual factors in the previous section, a standardization-customization decision is, however, based on the collective influence of all these factors – including the complex interactions between these variables. In this section, we present how we built and tested a standardization-customization decision model that takes in to account the collective influence of the factors discovered through our managerial interviews. Developing and testing such an empirical decision model, based on our case study, is a significant step towards a generalizable theory of the software product line evolution. Our empirical model of the standardization-customization decision complements other specification-based, process-centric, and architecture-centric approaches prevalent in the software product line evolution research [8, 9, 18, 19]. Our approach specifically highlights the collective and interactive effects of both software complexity and economic or market-related factors on product line evolution.

Figure 4 shows a pictorial representation of our standardization-customization decision model with the standardization-customization decision on the right-hand side and the variables representing our major hypotheses on the left-hand side. We also introduce three control variables in our model: size of the change request, customer influence, and the product version. We explain each of these in the following sub-sections.

4.1. Output Variable: Standardized or Customized Solution.

This output variable captures the standardization-customization decision that Measuretronics has to make on a customer's request. This is essentially a binary choice: to process the customer's request as a standard product feature in a future release (coded as 1), or as a separate customization project (coded as 0).

4.2. Input Variables

Installed-Base Impact. Whenever a change request is processed by Measuretronics, a survey is sent out to all customers to check if the requested change will be of any use to them. This variable gives the % of installed-base customers who indicated that the proposed change would be of use to them. If the proposed change is implemented in a future release, customers will have to install a patch even if they don't intend to use the new feature. Hence customers do not have any incentive to blindly accept any changes proposed by Measuretronics. This variable ranges from 0-100.

Standards Constraint. Upon the receipt of a customer request, Measuretronics verifies whether the request violates the standard Bluetooth Protocol standard at the time when the request was logged. We employed an indicator variable coded as 1 if the customer request violates the current Bluetooth protocol; 0 otherwise.

Compatibility Constraint. For every customer request that is received, Measuretronics also assesses whether the proposed feature will cause incompatibility with previous releases of the product, i.e., violate backward compatibility. We created an indicator variable coded as 1 if the proposed change request will cause incompatibility with previous public releases; 0 otherwise.

Market Share. This variable captures the market share (in %) of Measuretronics in the Bluetooth test product segment at the time when a customer request was logged. We obtained this variable from the product management division of Measuretronics. This variable ranges from 0 to 100.

4.3. Control Variables

Change Size & Nature of Change. We coded the nature of change based on our description in Section 3.4. We measure the extent of the change needed to fulfill a customer's request using the number of function points that are required to be developed or modified in order to implement the change request. In our data set we encountered situations where the function point data for a change request was zero. This is because some of the change requests only involved changing configuration settings and no addition or deletion of actual software code.

Customer Influence. A customer raising a change request with Measuretronics is required to indicate the importance of the request. This variable is coded on a Likert scale ranging from 1 to 5 and measures the perceived importance of the requested feature for a customer's business. The values of the scale were as following:

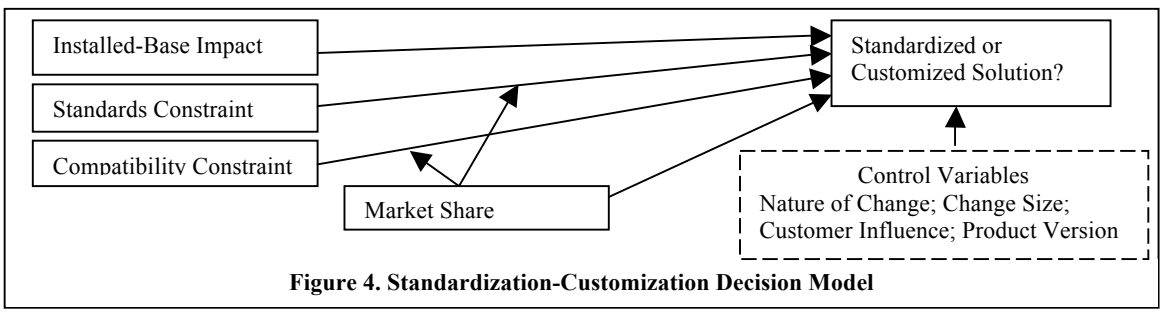

Figure 4. Standardization-Customization Decision Model

(1) The change is preferred by the customer but is not essential for day-to-day productive operations of the customer. The customer does not intend to be a test partner for implementing the change;
(2) The change is preferred by the customer but is not essential for day-to-day productive operations of the customer. The customer agrees to be a test partner for implementing the change;
(3) The change is needed for the day-to-day operations of the customer. The customer is willing to accept roundabout or alternate ways of implementing the feature and does not intend to be a pilot and test partner for the change. The customer cannot allocate resources to simulate a production environment;
(4) The change is needed for the day-to-day operations of the customer. The customer is NOT willing to accept roundabout or alternate ways of implementing the feature but can wait until the feature is released in future versions. The customer is willing to simulate a test production environment;
(5) The feature requested is essential and time critical. The customer is willing to allocate resources for piloting. The customer might stop using the vendor's product if the feature is not made available.

Product Version. There were 56 different versions of the packaged software product line. Hence this variable ranges from 1 to 56.

4.4. Interaction Effects

Our standardization-customization decision model also hypothesizes two interaction effects involving the market share of the product at the time when a customer request is logged. We expect that the market share of the product influenced the way Measuretronics made standardization-customization decisions especially when faced with decisions that involve violating compatibility with prior versions or any industry recommended Bluetooth standards.

5. VALIDATING THE STANDARDIZATION-CUSTOMIZATION DECISION MODEL

In this section we statistically test our proposed standardization-customization decision model using the data we collected through our case study. The empirical formulation of our standardization-customization decision model is shown in Equation 1:

Probability (Standardized or Customized Solution?) =

$$\Phi\ [\alpha_0 + \alpha_1 * \text{Installed-base impact} + \alpha_2 * \text{Compatibility constraint} + \alpha_3 * \text{Standards constraint} + \alpha_4 * \text{Market share} + \alpha_5 * \text{Customer influence} + \alpha_6 * \text{Market share} * \text{Compatibility constraint} + \alpha_7 * \text{Market share} * \text{Standards constraint} + \alpha_8 * \text{Change size} + \alpha_9 * \text{Version} + \alpha_{10} * \text{Nature of change} + \delta]$$

…(Equation-1)

The summary statistics of the data collected is shown in Table 1. Since the dependent variable (standardization-customization decision) is a binary choice, we used a probit regression model for estimating Equation 1. The results of the probit regression are presented in Table 2. The statistical test results indicate that our hypothesized standardization-customization decision model is *valid* – note the highly significant p-value for the model significance F-test in Table 2. In probit regression, the coefficients of the variables (α_0 - α_{10}) indicate an increase or decrease of log odds (probit score) of the dependant variable in standard deviations units – which is harder to interpret. For easier interpretation, we converted the probit effects to probability effects by calculating the effect of one unit increase in an independent variable on the probability of the vendor making a standardization decision holding all other variables at their mean levels. Table 3 shows these easier to interpret probability effects.

Robustness Checks. We performed a variety of robustness tests, including the influence of outliers (using Cook's distance), multi-collinearity effects (variance inflation factors and condition numbers), heteroskedacity checks (robust errors) to confirm the validity of our empirical results, and did not find any problems.

5.1. Decision Model Results

An analysis of the decision model results presented in Table 3 provides the following insights:

1. **Effect of Market Share:** A vendor with high market share (monopoly) in a heterogeneous demand environment will respond to customer change requests more through custom solutions than through product standardization. Specifically, we see that for every 1% increase in market share of its product, there is a decrease of 2% in the probability that Measuretronics would provide a standardized product offering for its diversified customer base (refer to coefficient α_4 in Table 3).

2. **Role of Standards:** When a customer requests a feature that violates industry standards, then the vendor is likely to fulfill the need only through a customized solution. This is true even when the market share of the vendor is high (monopoly). Specifically we find that a customer request that violates standards has a 61% lesser chance of getting accepted as a standard feature (See coefficient α_3 in Table 3).

3. **Role of Compatibility Needs**: When a customer requests a feature that is not compatible with prior versions of the vendor's product, the vendor is likely to fulfill the need through a customized solution. However, when the market share of the vendor is high (monopoly), the vendor might consider standardizing a feature even when presented with compatibility constraints. Holding all other variables at their mean level, we find that customer requests that have a compatibility constraint have 52% lower probability for getting standardized in a packaged software product (See coefficient α_2 in Table 3). However in the presence of higher market share, the probability of a customer request that has compatibility constraints getting accepted in to the standard version of the product increases by 25% (See coefficient α_6 in Table 3).

4. **Needs of Customers:** We observe that customers who declare a feature as essential to their operations have a significantly higher probability (+27%) of having their requests standardized (See coefficient α_5 in Table 3).

6. PRODUCT VARIABILITY COST MODEL

Costs of standardization are typically recovered through the product pricing mechanisms across several product releases common to all customers. However, costs involved in product line variability or customized solutions are typically borne by specific customers who, in turn, typically make purchasing decisions based on the product line variability pricing schemes. Hence it is important for Measuretronics to provide reasonably accurate and reliable cost estimates for product line variability (i.e., customized services). While several models of general cost estimation exist in the software engineering literature [4], to the best of our knowledge,

existing cost estimation models do not specifically address the costs involved in software product line variability by specifically taking in to account the customization lifecycle of a packaged software product. In this section, we develop and test a customization cost estimation model specifically suited for packaged product line variability. The empirical formulation of our cost estimation model is presented in Equation 2. Figure 5 shows a pictorial representation of our proposed customization cost estimation model. We explain each of these components of our model in the following sub sections.

Table 1. Decision Model Summary Statistics

Variable	Unit	Mean	Std. Dev.	Min	Max
Standardized or Customized Solution?	1 or 0 (decision)	0.60	0.49	0	1
Installed-Base Impact	%	43.64	27.65	5	100
Compatibility Constraint	1 or 0 (Present or absent)	0.47	0.50	0	1
Standards Constraint	1 or 0 (Present or absent)	0.29	0.46	0	1
Market Share	%	24.31	21.44	10	60
Customer Influence	Likert Scale	3.95	0.96	1	5
Market Share X Compatibility Constraint	(Interaction variable)	12.25	19.94	0	60
Market Share X Standards Constraint	(Interaction variable)	8.09	17.62	0	60
Change Size	Function Points	62.73	64.47	0	292
Nature of change	Incremental (0) or Architectural (1)	0.49	0.49	0	1
Version	Unit less number	28.22	16.93	1	56

$$\text{Customization Cost} = \beta_0 + \beta_1 * \text{Functional Complexity} + \beta_2 * \text{Structural Complexity} + \beta_3 * \text{Compatibility Constraint} + \beta_4 * \text{Customer Knowledge} + \beta_5 * \text{Quality} + \beta_6 * \text{Personnel Experience} + \beta_7 * \text{Size} + \beta_8 * \text{Version} + \varepsilon \quad \text{…(Equation-2)}$$

Product Variability Costs. We measure product variability costs in terms of the customization cost, i.e., the total person hours spent in the customization project for a customer. The effort spent for fulfilling customer requests at Measuretronics was tracked through a project planning and employee resource planning software. We extracted the relevant information from this system using the customer request id. The information was later verified by the project leaders of the customization teams.

Table 2. Decision Model Results

Variable		Coefficient	P-Value
Installed-Base Impact	α_1	**-0.019****	0.007
Compatibility Constraint	α_2	**-1.54****	0.003
Standards Constraint	α_3	**-1.88****	0.001
Market Share	α_4	**-0.06****	0.000
Customer Influence	α_5	**0.77****	0.000
Interaction effect: Market Share X Compatibility Constraint	α_6	**0.035***	0.016
Interaction effect: Market Share X Standards Constraint	α_7	-0.003	0.821
Change Size	α_8	**-0.008****	0.001
Version	α_9	0.018	0.290
Nature of change	α_{10}	**3.507***	0.010
Constant	α_0	0.59	0.454
Number of Observations = 154 Model significance test = **109.72**** ; P-Value= 0.000			

Table 3. Probability of standardizing a feature

Variable		Effect of one unit change on the probability to standardize a customer request
Installed-Base Impact	α_1	**-0.007****
Compatibility Constraint	α_2	**-0.52****
Standards Constraint	α_3	**-0.61****
Market Share	α_4	**-0.02****
Customer Influence	α_5	**0.27****
Interaction effect: Market Share X Compatibility Constraint	α_6	**0.25***
Interaction effect: Market Share X Standards Constraint	α_7	-0.02
Change Size	α_8	**-0.003****
Version	α_9	0.007
Nature of Change	α_{10}	**-0.3***

Note: For Tables 2 & 3, results significant at 5% are indicated by *; results significant at 1% are indicated by **. Other values, which are not in bold, are not statistically significant. We use a two-tailed hypothesis test.

Functional Complexity. Past research has classified functional complexity in to logical and data complexity [1]. Logical

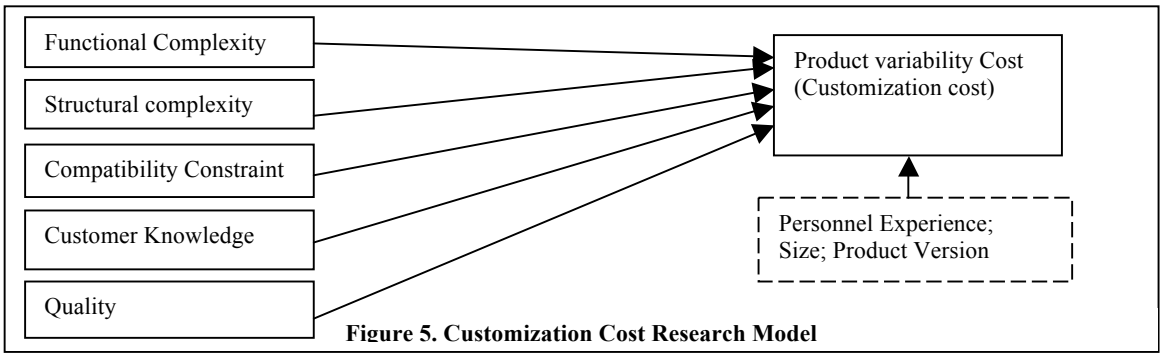

Figure 5. Customization Cost Research Model

complexity is the measure of complexity of decision-making and algorithmic logic in a program. Data complexity is a measure of the data processing complexity of a program. In a system software package such as the one involved in this research, logic and algorithmic functions dominate a program rather than the data processing functionality. Unlike enterprise applications such as ERP or CRM software, the Bluetooth protocol analyzer package that we studied had very minimal data processing capabilities. Thus, we used a logical complexity measure, McCabe's number [15], as the functional complexity of the software. McCabe's number measures the complexity of code based on the number of edges, the number of nodes and the number of connected components in a program flow graph. This measure is thus an indication of the number of decision trees available in the system. McCabe's complexity number is given by $V = e - n + 2p$. (e is the number of edges, n is the number of nodes, *and p is the* number of predicates in the program flow graphs).

Structural Complexity. Structural complexity results from the cross-references in the individual components of the packaged application. Past empirical software engineering studies have acknowledged the influence of coupling and cohesion on maintenance performance [1, 6]. Our product was designed using the object oriented programming paradigm. We, thus, used the object oriented metric from the Chidamber-Kemerer metric suit, Coupling Between Objects (CBO), as the measure of structural complexity [7].

Compatibility Constraint. As mentioned in Section 4.4, for every customer request that is received, Measuretronics also assesses whether the proposed feature will cause incompatibility with previous releases of the product, i.e., violate backward compatibility. We created an indicator variable coded as 1 (0 otherwise) if the proposed change request will cause incompatibility with previous public releases.

Customer Knowledge. At Measuretronics, any prior experience with a customer was documented by the product management team in the form of structured notes stored in the knowledge management repository for future engineering reference and pursuing marketing leads. We gathered the customer identification number from the customer requests and used this to retrieve all the customer notes that the product management and developers had stored till date. We use the total number of such structured notes available for a customer as an input variable in the model.

Quality. We measure quality as the number of errors that have been reported for the components that are relevant to the customers' change request. It is important to accommodate quality in our cost estimation model because more error prone components might require more time to test than less error prone modules.

Personnel Experience. We gathered information about the technical experience and Bluetooth domain experience of the personnel who participated in processing the customization request from the user. This variable was calculated as the average of the technical and domain experience (in years) of the team that processed the customization request.

Size. We measured the scope of a change request using the number of function points that are required to be developed or modified in order to implement the change request.

Product Version. There were 56 different releases of the packaged software product. Hence this variable ranges from 1 to 56.

6.1. Product Variability Cost Model Results

We used an ordinary least-squares regression to estimate Equation 2. Table 4 presents the summary statistics for our model. The results of the regression are presented in Table 5. The model specified in equation 2 is highly significant and robust (Notice the highly significant P-value for the model test in Table 5). The explanatory or predictive power of the model is high and the overall mean Magnitude of Relative Error (MRE) for the model was 19%, which is lower than those reported for other estimation models [6]. Our model thus accurately predicts the man-hours (to within 19% of the actual value on average) needed for fulfilling a particular customization solution. An analysis of the results shown in Table 5 reveals the following insights:

Effect of functional complexity: Higher levels of functional complexity are associated with higher customization costs. Specifically we find that for every unit increase in McCabe's number, the effort needed to satisfy a customization change request goes up by fifteen minutes if handled by a team with an average of five years of experience holding other variables at their mean levels.

Effect of structural complexity: Higher levels of structural complexity are associated with higher customization costs. In our analysis, we find that for every unit increase in the Coupling-Between-Object metric, customization costs go up by about 4%, holding everything else at their mean levels (see β_2 in Table 5).

Compatibility constraints: Customer requests that require solutions that are incompatible with other publicly released versions of the product will usually result in higher customization costs. On average, we found that developing customized solutions with compatibility constraints required 175% more effort than those that did not face incompatibility problems (see β_3 in Table 5).

Customer specific knowledge: Higher levels of customer specific knowledge available with a vendor usually results in a small, yet, significantly lower customization costs for that customer's requests (an average of 0.5% reduction in costs; see β_4 in Table 5). We believe this was due to the structured way in which knowledge was organized at our research site. All customer specific information was organized in the form of similar sized notes with key references to customer request ids, product feature ids, and stored in a central repository that was accessible to all personnel. This way of organizing knowledge helped to reduce costs by ensuring uniformity in understanding customer needs between personnel involved in development and maintenance activities of the product.

Effect of Quality: Error prone components from prior releases are associated with higher customization cost. Our results show that a unit increase in the incidence of errors in the modules being modified for the custom solution increases customization cost by about 11% (see β_5 in Table 5).

Effect of Experienced Personnel: We find that higher technical and domain experience of the development team involved in customization is associated with lowered customization costs. Our results indicate that an increase in one year of average team experience lowers customization costs by 28% (see β_6 in Table 5).

Table 4. Summary Statistics for Customization Cost Model

Variable	Unit	Mean	Std. Dev.	Min	Max
Customization Cost	Person-hours	287.35	277.52	8	1000
Quality	Count of errors	8.24	7.06	1	29
Structural Complexity	Coupling-Between-Objects	18.53	22.77	0	105
Functional Complexity	McCabe's number	174.94	312.23	1	1792
Compatibility Constraint	Indicator – (1- present; 0- absent)	0.41	0.50	0	1
Customer Knowledge	Count of similar sized notes	109.29	134.27	0	695
Size	Function Points	1211.28	2323.21	54	15688
Personnel Experience	Years	5.21	1.75	2.5	8
Version	Count	28.25	15.52	2	55

Table 5. Customization Cost Results

Variable		Coefficient.	P-value
Functional Complexity	β_1	**0.25****	0.035
Structural Complexity	β_2	**3.57****	0.017
Compatibility Constraint	β_3	**175.59*****	0.004
Customer Knowledge	β_4	**-0.53****	0.036
Quality	β_5	**10.88****	0.024
Personnel Experience	β_6	**-28.08***	0.073
Size	β_7	0.022	0.123
Version	β_8	-1.97	0.302
constant	β_0	**247.95****	0.024
Number of Observations = 51 Model Significance test F-stat (8, 42) = **9.28***, P-value = 0.000 Adj.R-Squared = 56.99%; Mean MRE = 19%			

Note: Results significant at 5% are indicated by **; results significant at 1% are indicated by ***. Other values, which are not in bold, are not statistically significant. We use a two-tailed hypothesis test (i.e., we did not assume any positive or negative direction of the result while testing).

7. DISCUSSION

Putting Lessons in to Practice. While our results discussed in previous sections were specific to this case study, the lessons learnt can be cautiously generalized to the broader product line evolution context – eventually paving way for a generalized theory of software product line evolution through replications and confirmation in other contexts. We summarize our recommendations on putting our lessons learnt to practice in the following steps:

1. Costs involved in product commonalities (standardization) and product variabilities (customization) need to be rigorously calculated and debated. Our empirical models are a step forward in this direction.
2. Product line governance needs to include collective product decision-making aligning the incentives for solving engineering problems and meeting market opportunities. Our case study illustrated such a collective decision making process in one firm.
3. Benefits of pleasing customers through customized solutions need to be balanced with the risks of violating technical standards and increasing complexity due to product line variabilities. These risks need to be appropriately transferred to customers (though pricing as shown in our product variability cost model) or through controlled product feature obsolescence.
4. Appropriate investments for organizational learning such as knowledge management systems and resource rotation between development and maintenance improves the general capability of the product management team, and prepares them for collective decision-making.

We discuss the broader implications brought out by the lessons from our case study and empirical models in the rest of this section.

Collective Product Decisions. Making software product management decisions is challenging because managers have to simultaneously consider a multitude of factors including technical complexity, economic incentives, and market forces. Our case study data enumerates the influence of these factors on the evolution of a complex software product. The generalizable standardization-customization decision model and product variability cost model we have illustrated using the case study data can be used to add rigor to the software product management decision making process. This would enable managers to systematically estimate the implications of complexity and the nature of change requested by customers, along with other market considerations such as market share and competition, to make optimal software product management decisions. Overall, this research fills an important gap in the toolbox of software managers who are involved in decision making of software product lines.

Timing the Product Feature Obsolescence. We find that a universal policy to enforce compatibility of standard product releases is not an optimal strategy for packaged software product lines. We found that standardizing an incompatible feature in to a packaged software product, especially when a customer requested the feature and when the vendor owns a high market share, could be a winning strategy. Under these specific conditions, standardizing incompatible features in future versions of the product, instead of fulfilling such requests through product variabilities (customized solutions), could motivate the larger installed-base to purchase newer product versions, and help product management teams to plan for controlled obsolescence of older product versions.

Economic Incentives of Managing Complexity. A packaged software product vendor would typically hold the core source components of a product platform for multiple years. We demonstrate that there is an economic incentive to reduce structural complexity as the product matures over the years. Measuring and acting on the impact of complexity during the early design phase would save costs later in the product lifecycle because expenses related to providing hot fixes throughout different versions of the product are minimized. One way to reduce the impact of structural complexity on product variability costs would be to separate out design components according to the holistic requirement needs of consumers, and modularize these components in different packages [20]. This would reduce the cross-referential changes needed for more volatile business components, and thus minimize costs of developing and testing customized solutions.

Invest in Organizational Memory. We learned from our research that identifying and resolving compatibility constraints in downstream activities, such as customization, is a difficult task. This difficulty multiplies in magnitude when the personnel involved in developing the customized solutions are different from the original system designers and application developers (the current situation at our research site). The probability of critical design knowledge being lost between these different teams is quite high and special effort is needed to synchronize the understanding of the product between the product management team, application developers, and custom project developers. Overall, our experience at this research site reinforces the benefits that teams can derive by using structured knowledge management practices in software development and product management.

8. RELATED WORK

We draw inspiration from four streams of literature for this study. Our modeling approach in this study is based on economic modeling of software development and maintenance activities published in the software engineering economics literature [1, 12]. We extend prior work in this area by considering the packaged software standardization-customization decision and estimation of customization costs. We draw ideas from the product development literature to identify the factors that influence the standardization and customization of software products. The product development literature covers perspectives from diverse fields of marketing, organization science, engineering design, and operations [5, 11, 13]. Our third literature reference is the software product line stream of research [8]. Prior research in requirements engineering [14, 17] and software change management [16] also serve as a motivating foundation for this study. Our study, bridges the approaches from these four different research streams in an attempt to advance towards a generalizable theory of software product line evolution.

9. CONCLUSION

In this paper, we first showed the factors that a packaged software vendor considers when deciding whether to satisfy a customer's change request by either a) offering a customized solution to the individual customer, or b) offering a standardized solution by releasing a new feature in a future version of the product. We then developed a predictive decision model that allows managers to decide whether customization or standardization is the appropriate response to a customer change request. In addition, we built a product variability cost estimation model that allows managers to accurately predict the cost of a customized solution. Our research, by accounting for intermeshed market effects and complexity dynamics in software products, makes an important advancement towards a generalizable theory of software product line evolution.

10. REFERENCES

[1] R. D. Banker and S. A. Slaughter, "The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement," *Information Systems Research,* vol. 11, pp. 0219-0240, 2000.

[2] Bluetooth-SIG, "http://www.Bluetooth.Com/Bluetooth/Sig/History_of_the_Sig.htm," accessed on 5-Sep-09

[3] Bluetooth, "http://www.Bluetooth.Com/Bluetooth/Technology/" accessed on 5-Sep-09

[4] B. Boehm, C. Abts, and S. Chulani, " Software Development Cost Estimation Approaches — a Survey," *Annals of Software Engineering,* vol. 10, pp. 177-205, 2000.

[5] S. L. Brown and K. M. Eisenhardt, "Product Development: Past Research, Present Findings, and Future Directions," *Academy of Management Review,* vol. 20, pp. 343-378, 1995.

[6] D. Card and R. Glass, *Measuring Software Design Quality*. Englewood Cliffs, NJ: Prentice Hall, 1990.

[7] S. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering,* vol. 20, pp. 476-493, 1994.

[8] P. Clements, C. Gacek, P. Knauber, and K. Schmid, "Successful Software Product Line Development in a Small Organization," in *Software Product Lines: Practices and Patterns*, P. Clements and L. Northrop, Eds.: Addison Wesley Longman, 2001.

[9] G. C. Gannod and R. R. Lutz, "An Approach to Architectural Analysis of Product Lines," in *22nd International Conference on Software Engineering*, Limerick, Ireland, 2000.

[10] IDC, "Wordlwide Software Market 2009-2013 Forecast," IDC2298780, 2009.

[11] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software,* vol. 19, pp. 58-65, 2002.

[12] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay, "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science,* vol. 46, pp. 0745-0759, 2000.

[13] V. Krishnan and K. T. Ulrich, "Product Development Decisions: A Review of the Literature," *Management Science,* vol. 47, pp. 0001-0021, 2001.

[14] T. v. d. Maßen and H. Lichter, "Requiline: A Requirements Engineering Tool for Software Product Lines," *LNCS,* vol. 3014, pp. 168-180, 2004.

[15] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering,* vol. SE-2, pp. 308-320, 1976.

[16] K. Mohan and B. Ramesh, "Change Management Patterns in Software Product Lines," *Communications of the ACM,* vol. 49, pp. 68-72, 2006.

[17] B. Regnell, P. Beremark, and O. Eklundh, "A Market-Driven Requirements Engineering Process: Results from an Industrial Process Improvement Programme," *Requirements Engineering,* vol. 3, pp. 121-129, 1998.

[18] K. Schmid, "A Comprehensive Product Line Scoping Approach and Its Validation," in *24th International Conference on Software Engineering*, Orlando, Florida, 2002.

[19] M. Svahnberg and J. Bosch, "Evolution in Software Product Lines: Two Cases," *Journal of Software Maintenance: Research and Practice,* vol. 11, pp. 391-422, 1999.

[20] K. T. Ulrich and D. J. Ellison, "Holistic Customer Requirements and the Design-Select Decision," *Management Science,* vol. 45, pp. 0641-0658, 1999.