Qualitative SIMPLE

John D. McGregor, Clemson University and Luminary Software LLC, U.S.A.

Abstract

SIMPLE, Structured Intuitive Model of Product Line Economics, provides a technique for modeling the costs and benefits of a software product line over its life time. Several scenarios of product line development have been identified and the equations developed for those scenarios. The SIMPLE technique was intended for creating quantitative estimates. In this issue of Strategic Software Engineering I want to discuss a qualitative approach to building SIMPLE arguments for use when obtaining exact numbers may be impractical.

1   INTRODUCTION

Teams planning to adopt the software product line strategy often must develop a series of justifications and evaluations before receiving approval for full adoption. The initial justifications are usually intended to convince a manager to approve funds to conduct a more detailed study. Usually obtaining exact numeric data is resource intensive and outside the scope of an initial investigation. On the other hand, the initial investigation serves no purpose if it does not analyze information of sufficient importance to make a believeable conclusion.

In lieu of specific data, the study team must bring their personal experiences to bear on the problem. That means the study group should be as diverse as the skills required for the product line strategy. The Software Engineering Institute's Framework for Product Line Practice describes 29 practice areas that define the software product line strategy. That does not mean the team needs 29 people. The practice areas are divided into software engineering, technical management, and organizational management. Usually an initial team of 5 to 7 people can cover these areas adequately, when they can quickly call upon other experts in the organization.

The study team makes models to support their investigation. There are several ways in which a qualitative model can be constructed. They may define the models using text or they may create a more formal model using some notation. Exact values can be replaced by ranges of values. Precise comparisons can be replaced by rankings and constraints. We can go further and use the elements of qualitative reasoning to build models that can be animated and reveal basic possibilities.

In this paper I will explore a technique for building economic models of software product line organizations. The technique was the result of a collaboration of several people and is being extended and nutured by the Software Engineering Institute (SEI). I will explain the basic model and then I will illustrate a variety of ways that it can be used. I will show fragments of a couple of tools that can be used to support decisions about product line adoption.

2   SIMPLE

The Structured Intuitive Model of Product Line Economics (SIMPLE) provides a set of functions that account for the expenses and benefits of building the product line and operating the product line organization [Clements 05]. We took the approach of using cost and benefit functions without defining a specific implementation of the cost functions to allow the maximum flexibility for the modeler. For a specific situation, often defined by scenarios, the modeler determines the best implementation for the function.

A basic cost model consists of the set of four cost functions given in the expression (In equation (1) the *n* indicates the number of products.):

$$C_{org}(\Box) + C_{cab}(\Box) + \sum_{i=1}^{n} \left(C_{unique_i}(\Box) + C_{reuse_i}(\Box) \right) \tag{1}$$

C_{org}: This is the cost of the product line organization. This usually can be divided into two parts: the cost of start-up and the cost of on-going operations. *Example: The cost of training personnel on new processes and new tools.*

C_{cab} : This is the total cost of the core asset base over the life of the product line. Remember the core assets are those items that are used across multiple products in the product line. This cost function estimates the cost over all 29 practice areas. It includes the labor to construct the core assets plus other costs, such as the cost of tools to create the asset and tools and activities to test the asset. C_{cab} represents both the initial cost of development and the cost of maintenance. *Example: The cost of developing the product line software architecture.*

C_{unique_i}: This is the cost of the unique portion of a particular product. The i subscript indexes the set of products. Each product will have a different value since each has unique content. *Example: The cost of a more exhaustive search algorithm than required for the other products in the product line.*

C_{reuse_i} : This is the total cost of reusing the core assets that are being composed to form the product. The attached processes of the assets are intended to reduce this cost by providing a "user's manual" for the core asset. *Example: The cost for learning the meaning and range for each parameter on a component's constructor.*
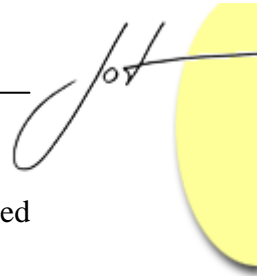
The following addiitonal functions are in the SIMPLE bag of tricks and can be used to construct various views of the costs and benefits of product line practices.

Ben_j: These are the anticipated benefits from using the product line approach. They are not reduced costs, which would be reflected in the cost functions. A benefit is a positive or negative impact on factors other than costs, such as a new or increased revenue stream. *Example: An increase in quality might lead to increased sales that would not have happened without the product line strategy.*

C_{prod}: This is the cost of building a product in a non-product line environment. It is typically used to construct a comparison argument comparing the current practice and a product line approach. *Example: This cost is computed using the organization's existing techniques for development costs.*

The functions discussed in this section have appeared in a variety of publications with differing numbers of parameters. These include the following:

$product_i$: This parameter refers to the product definition. It may be the set of requirements for the product or it may be an exact product name if the objective is to convert an existing set of products into a product line.

t_i: This parameter is time. The time may take one of several forms. It might be actual calendar time if the model is defining actual production periods or it may be a relative time, such as first to be produced, if the intent is to focus on conflicts and opportunities.

$C_{unique_i}(product_i, t_i)$ represents the cost of the unique portion of the ith product whose specification is given by $product_i$ and to be constructed at time t_i.

Using SIMPLE

One approach to using SIMPLE is to build cost estimates using the Framework for Product Line Practices [SEI 08]. Figure 1 shows a portion of a spreadsheet for computing estimates for product line costs and benefits. For each practice area, the cost of that practice area is allocated across the four cost functions. For an early estimate in an initial study, having this breakdown can help the manager focus on an individual item at a time.

To compensate for the quality of the estimates of costs and benefits on the individual practices we use a Monte Carlo simulation. The columns to the right of the cost estimates define intervals over which the estimated values should be jittered. The intervals are symmetric so a single value is used. The more confidence we have in the correctness of the estimate the smaller we make the interval.

Input Values (input)

					Min	Max			
NPV Year 0									
-2.904207482									
Year			0						
Corg			0.97499						
Ccab			1.92072						
Cunique			1.55707						
Creuse			0.13936						
n			0						
t									
Cprod			2.70037						
Percent Unique			0.5						
Interest Rate			0.05						
Profit per Product			4.81044		4.25	5.75			
Cprod			3		2.55	3.45			

Practice Areas

Software Engineering	Corg	Ccab	Cuniq	Creuse		Corg +/-	Ccab+/-	Cuniq+/-	Creuse+/-
Architecture Definition	X	0.5	X	X		X	0.02	X	X
Architecture Evaluation	X	0.3	X	X		X	0.02	X	X
Component Development	X	0.1	0.1	X		X	0.01	0.01	X
COTS Utilization	X	0.1	0.1	X		X	0.01	0.01	X
Mining Existing Assets	X	0.1	0.1	X		X	0.02	0.01	X
Requirements Engineering	X	0.1	0.1	X		X	0.03	0.01	X
Software System Integration	X	0.1	0.1	0.05		X	0.03	0.01	0.001
Testing	X	0.1	0.1	X		X	0.02	0.02	X
Understanding Relevant Domains	X	0.1	X	X		X	0.01	X	X

Figure 1 - SIMPLE Spreadsheet

The simulation is run several thousand times. Using a normal distribution the mean value is used as the estimate. The resulting values can be analyzed in a number of ways.

Note the cell labeled "Year" in the spreadsheet. In this engagement we were creating estimates on a yearly basis. Each year is represented by a separate sheet. This longitudinal look provides the opportunity to play what-if with the distribution of activities. To make this longitudinal view useful, the Net Present Value (NPV) is used to adjust costs in the future to the common base of costs today.
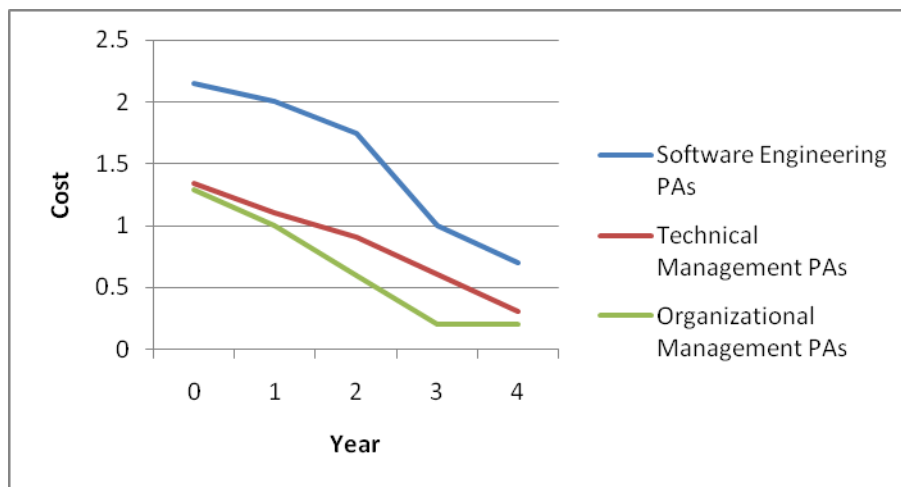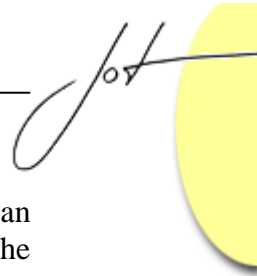

Figure 2 - Three essential activities by year

Figure 2 shows a graph that captures the cost behavior of the practice areas over an assumed 4 year life cycle of the product line. Figure 3 shows a graph that captures the behavior of the four cost functions over the 4 year life cycle. The increase in C_{cab} in year 3 is the result of a refresh of core assets. As expected, the line for C_{unique} reflects no consistent trend. Some products will have more unique features than others and will not necessarily show up at any specific point in the time line. C_{org} slowly declines as processes become routine, and in some cases automated. C_{reuse} trends down slowly as the product development staff becomes more familiar with the assets.
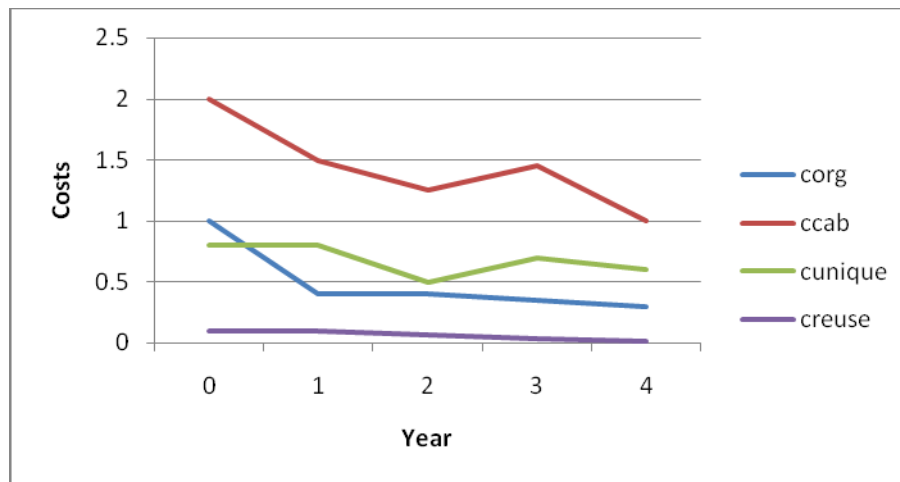

Figure 3 - Cost functions by year

Relationships

There are relationships among various costs that the modeler should keep in mind in making estimates for the cost functions. Some of these relationships directly involve these functions and some are more indirect depending on outside influence. Here are some we have identified over time.

As *n* increases there is probably an increase in the cost of building the core assets. Even though C_{cab} is listed as a fixed cost outside the summation, there is an influence.

As *n* increases there is probably an increase in the organizational costs. There may be more support staff needed and other related costs. Even though Corg is list as a fixed cost outside the summation, there is an influence.

As C_{cab} increases because of increased automation, the C_{reuse} probably decreases. Automated assets will be harder to construct but easier to use.

As the core asset base evolves toward a configurable platform [Bosch 02], C_{unique} will decrease.

The smaller the development team the more rapidly C_{reuse} will decrease as the developers become familiar with the core assets.
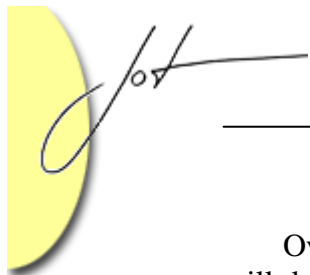
Over time some unique pieces will migrate and be promoted to core assets. C_{unique} will decrease and C_{cab} will increase but the increase should be offset by multiple uses of the new core asset. The net result of a promotion should be a decrease in total costs.

3   QUALITATIVE SIMPLE

During the initial stages of a study to determine whether or not to adopt the product line strategy it is often sufficient to consider relative costs rather than exact, total costs for each scenario. A relative comparison of costs among scenarios allows readers to focus on relationships among costs and what will happen to these costs as changes occur in the product line.

I often create a comparison table in which multiple options can be described in terms relative to each other. Table 1 shows a very small example of such a table. The users of this table evaluate the scenarios for each cost and benefit and judge which scenario is best for the current state of the product line organization. In Table 1 scenario #1 has more advantages.

Scenario #	C_{org}	C_{cab}	C_{unique}	C_{reuse}	Ben
1-incremental	No reduction in costs.	Reduces up front cash needs	Can take advantage of opportunities to promote unique to core assets	No difference	Quicker response to unanticipated opportunities
2-proactive	Most expenses here will be upfront under any scenario.	May be easier to estimate since all expenses happen upfront.	No opportunity to identify additional commonality	No difference	Quicker response to anticipated opportunities

Table 1 - Qualitative comparison

The scenarios are usually more detailed than the ones in Table 1 and the best selection is harder to judge. Qualitative modeling is a more formal approach that provides the ability to evaluate conflicting forces more accurately at the expense of more time and effort to build the model. The model fragment in Figure 4 shows a fundamental structural relationship between a product and the product line: a product line contains products. Other fragments will relate core assets and unique pieces to the product line.
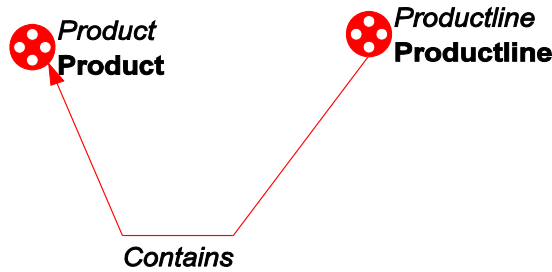
Figure 4 - Structural relationship

Qualitative reasoning provides a technique for modeling entities, quantities and interactions without numeric values. In Figure 5, a product and the core asset base are related via a "positive proportionality" (P+) between the number of products and the cost of the core asset base. This fragment models the fact that as the number of products is increasing the cost of the core asset base also increases. The cost is modeled using a quantity space in which a quantity is either increasing, decreasing, or constant. The number of products is modeled as a value that is positive with a positive derivative.
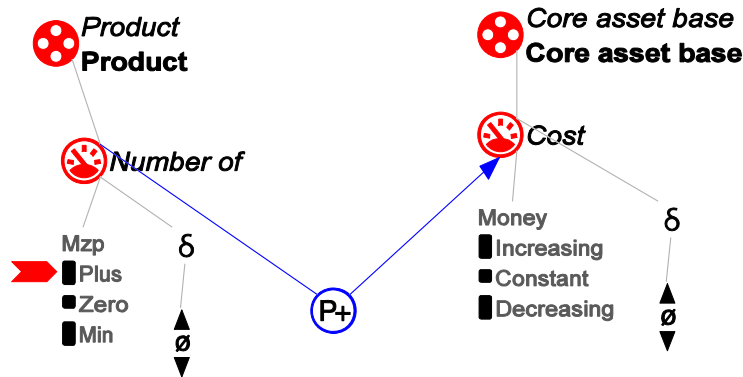

Figure 5 – Fragment of a qualitative model

Figure 5, 5, and 6 were created using Garp3 [Garp 08]. This tool also provides an animation tool that can trace all possible combinations of value states. Scenarios are constructed that identify a stimulus into the model and the initial conditions of the model when the stimulus is received. The animation propagates the stimulus and simulates the propagation of causal actions through the model. The simulation will not necessarily produce a single answer since qualitative arithmetic is ambiguous, but it can reduce the total number of possibilities. Figure 6 shows a scenario, in which the product line manager agent adds a product to the product line. The simulation algorithm will propagate this input and update changes to linked quantities using the arithmetic defined on the quantities.
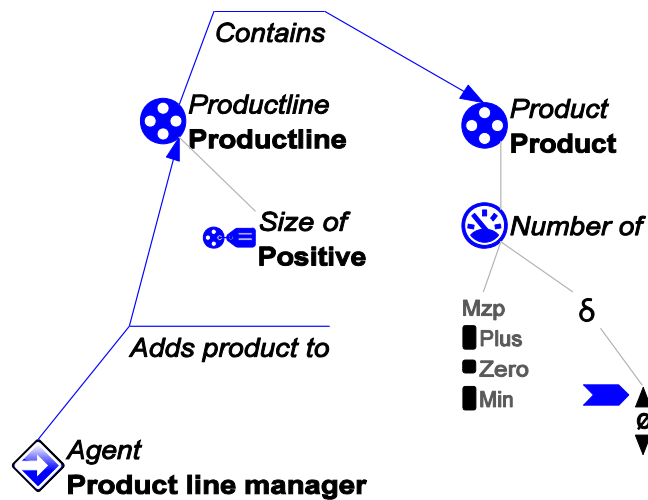
Contains

Productline
Productline

Product
Product

Size of
Positive

Number of

Adds product to

Mzp
Plus
Zero
Min

δ

ø

Agent
Product line manager

Figure 6 - Qualitative scenario

Notice that the "Number of" attribute of Product is measured on the scale of Plus, Zero, and Minus. Beside that scale the arrow is pointing to the upper pointing triangle indicating that the derivative of that quality is increasing. There is a reasoning algorithm for using quantities and their derivatives to further propagate scenario actions allowing a scenario to be evaluated across all the quantity spaces and narrowed to a few possible results.
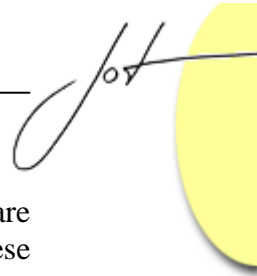
4   ROI

Many product line decisions, particularly the initial adoption decision, are investment decisions and many investment decisions are evaluated on the return that will be realized as a result of the investment. The standard formula for computing return on investment (ROI) is available from many sources including [Answers 08]:

$$\frac{benefit\ resulting\ from\ investment}{amount\ invested} \qquad (2)$$

One way to model the benefit from product line adoption is to compare the cost of building the systems in the existing way and building the systems in a product line manner. The Ben function may be included to more accurately reflect the product line benefits but these should only be benefits directly attributable to the product line approach. The ROI in a product line environment could be modeled as [Böckle 04]:

$$\frac{\sum_{i=1}^{n} C_{prod_i} - \sum_{j=1}^{m} Ben_j - \left((C_{org} + C_{cab} + \sum_{i=1}^{n}(C_{unique_i} + C_{reuse_i})) \right)}{C_{org} + C_{cab}} \qquad (3)$$

The ROI for a software product line will change over its life as additional investments are made to refresh core assets. In the next section we look briefly at modeling these invetsments.

5   OVER TIME

The context in which the product line is established changes with time. [Ganesan 06] discusses how a product line organization must refresh the core assets periodically. Essentially these costs are seen in the C_{cab} and C_{unique} functions.

The technologies, market information, and domain knowledge captured in core assets has a finite life time. In some domains that life time may be quite fast – a few months – or more slowly – a year or more. The SIMPLE model accommodates that life time by computing the model in segments where each segment is approximately the length of time before an asset needs to be refreshed. Remember that the uptick in year 3 for C_{cab} in Figure 3 reflected a revision of core assets.

Over time some of the artifacts that are initially viewed as unique will be promoted into the core asset base. This happens when a product team recognizes the need for the same functionality as has been used in another product. In the promotion to core asset, costs associated with the two occurrences in C_{unique} are moved to C_{cab}. C_{cab} also increases by the amount needed to revise the unique portion to make it reusable. The intent is that over time the reduction in C_{unique} will be greater than the increase in C_{cab}.

6   SUMMARY

The initial investigations in product line adoption are high-level, but still critical to the ultimate success of the product line. Decisions about which products to include in the software product line and how to structure the development of the products are economic decisions. These are strategic decisions that involve the commitment of large amounts of resources and may affect the organization for some time in the future. SIMPLE has been used in a number of situations to assist in this type of decision making.

SIMPLE helps an organization considering adopting the software product line strategy to build the justification using the costs and benefits related to the practices required for product line success. SIMPLE supports a variety of types of arguments. SIMPLE can support numeric models and qualitative models. The techniques in this article should help make strategically useful models and ultimately lead to correct decisions.

7 ACKNOWLEDGEMENTS

REFERENCES

[Answers 08] Answers.com. http://www.answers.com/topic/return-on-investment, 2008.

[Böckle 04] Günter Böckle, Paul Clements, John D. McGregor, Dirk Muthig, and Klaus Schmid. Calculating ROI for Software Product Lines, IEEE Software, 2004.

[Bosch 02] Jan Bosch. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization, Second Conference Software Product Line Conference, SPLC2, August 2002.

[Clements 05] Paul Clements, John D. McGregor, and Sholom G. Cohen. The Structured Intuitive Model for Product Line Economics (SIMPLE), Software Engineering Institute, CMU/SEI-2005-TR-003.

[Ganesan 06] Dharmalingam Ganesan, Dirk Mutig, and Kentaro Yoshimura. Predicting Return-on-Investment for Product Line Generations, 10th International Software Product Line Conference (SPLC'06) pp. 13-22, 2006.

[Garp 08] Garp3, http://hcs.science.uva.nl/QRM/, 2008.

[SEI 08] Software Engineering Institute, www.sei.cmu.edu/productlines, 2008.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.