

A Study on Monetary Cost Analysis for Product-Line Architectures

Arkadeb Ghosal and Alberto Sangiovanni-Vincentelli

University of California, Berkeley

Sri Kanajan

General Motors

Copyright © 2008 SAE International

ABSTRACT

We present a cost model that analyzes monetary costs for a product-line architecture to help the optimization of the architecture. The paper illustrates the usefulness of this methodology in a case study based upon the design exploration of a product-line architecture for an active safety system.

INTRODUCTION

There has been an explosion in automotive electronics related to infotainment, communication and control. In particular, electronic control has become a central feature within steering, gear shifting and braking. We are witnessing a shift from mechanical coupling to x-by-wire technology as well as option packaging to manufacture cars for targeted market segments using the same product-line. In this scenario, the design decisions on the electronic architecture of a product-line are of paramount importance in determining the overall appeal of a car. The metric used to evaluate architectural alternatives should be as objective as possible. At this time, monetary cost is a good estimation of the quality of the decisions made. We believe that a comprehensive/absolute monetary cost model that captures the influence of the different architectural degrees of freedom on cost elements within the architectural life cycle is necessary.

Several approaches to cost modeling were presented in literature. Technical Cost Modeling (TCM) [2] analyzes the primary cost drivers associated with the manufacture of a product by modeling the capabilities and constraints associated with the processes used. TCM creates models to understand the sensitivity of manufacturing operations to changes in design and processing decisions. The TCM methodology distinguishes between two types of costs: fixed and variable. Fixed costs of production, i.e. investments in tools, equipment and facilities necessary for production, do not scale in direct proportion with production volume. Variable cost

categories, like direct/indirect materials, labor and energy, are typically consumed at a relatively constant rate based on part mass, cycle times and processing efficiencies, and tend to exhibit little cost scaling with production volume. The use of TCM is solely for determining the difference in cost between two options and not at detailed cost modeling for estimated part pricing. The cost model proposed in [3] accounts for hardware design, software design and part cost of the components of an electronic architecture. The UML based cost model aims for cost pricing rather than focusing on sensitivity of alternative designs to design decisions. Another approach is discussed in [5] where an assumed product-line cost model is used within an optimization scheme to map functions to ECU's automatically to minimize cost. A comprehensive cost model for a single *Electronic Control System* (ECS) architecture instance exploration was proposed in [1]. While understanding the cost factors of a single architecture instance is certainly important, a cost model for a product-line architecture has by far a larger impact. Even if a product-line architecture consists of several architectures instances (one for each package in the product-line), the cost of the product-line is not a "straightforward" summation of the cost of the architecture instances. For example, the software development cost for a particular module should not be counted multiple times even if the module may be reused in the other packages of the same product-line.

The approach proposed in this paper extends the cost model for an architecture instance [1] to a product-line architecture. The contributions of this paper are as follows: (1) a formal methodology to capture product-line architecture alternatives, (2) a cost model technique to compare two product-line architecture alternatives on the basis of design and development, assembly and manufacturing costs, (3) a case study to show the use of such a model in evaluating design decision, and (4) the use of sensitivity analysis in analyzing the robustness of architecture alternatives in the presence of varying cost model parameters.



Documents

Document Sharing Hub

COST MODEL FOR ECS ARCHITECTURE INSTANCE

An Electronic Control System (ECS) architecture instance [1] consists of a set of interconnected hardware modules and a set of software modules that are executed on the hardware modules. A software module is a block of code e.g. a real-time task or an IO driver. A hardware module may include a set of components likes CPU, memory, network controllers, connectors, a smart sensor, a smart actuator, or a specialized peripheral. A hardware module is also characterized by the packaging it requires (possibly specified by the size of PCB and housing) and the position (in the vehicle) where the module would be placed. The interconnection among hardware modules is specified through harnessing components (e.g. cut-leads, connectors or splices) and bussed electrical center (BEC) components (e.g. relays fuses and circuit breakers).

This architecture description is derived based on interviews of domain experts and analysis of a number of architecture studies. The model may not be comprehensive; however it captures the key architectural design decisions to estimate important cost intermediates and the subsequent effect on relative cost of the ECS architecture alternatives. Next an overview of the cost model is presented (refer to [1] for a detailed formal model).

The cost model encompasses the factors that affect the life cycle cost of an ECS architecture instance. The life-cycle cost includes design and development cost, part-fabrication cost, assembly cost and in-service cost. Figure 1 shows the primary cost drivers integrated into the hub of an influence diagram and illustrates the underlying structure of the comprehensive cost model. Cost drivers are further categorized as (1) fixed (costs gets amortized over more units with increase in production volume and unit costs show economies of scale) vs. variable (cost remains constant on a per-unit basis regardless of production volume), and (2) internal (cost carried, incurred and driven by the OEM) and external (cost incurred by suppliers of ECS components).

The *design and development cost* includes the cost of specifying, engineering, and validating the design. Given the increasing OEM interest in the role of software development, design and development cost is further disaggregated into two sub-categories: software and hardware development cost.

Software Development Cost

The fixed cost for software includes IT infrastructure, software licenses and office space (both for OEM and suppliers) - all items that are relatively independent of

the architecture chosen. For both OEM and supplier, the variable cost, e.g., cost of engineering effort for coding, depends greatly on architecture design decisions. The model computes the variable cost based on the widely accepted COCOMO [4] model which computes software design effort based on module interface (e.g., number of inputs, outputs, inquiries, etc), and development constraints (like project and organization expertise).

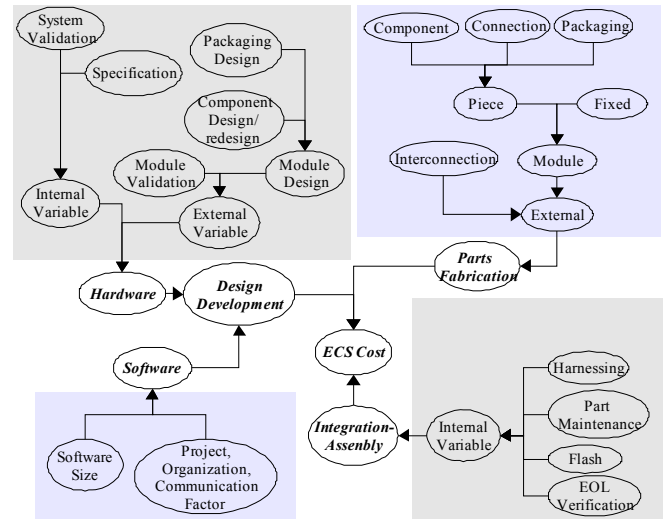


Figure 1: Cost model for ECS architecture instance

Hardware Development Cost

The hardware development cost accounts for the cost of specifying and designing the hardware and validating the operation of the integrated system. Fixed costs (internal and external) include items like IT infrastructure and personnel, tools, test benches, prototype set ups and management and are assumed relatively independent of the architecture decisions. The variable costs consist of OEM engineering activities, like total system specification, documentation and validation, summed with supplier engineering activities, such as design, development, and validation of the hardware modules.

The internal hardware design variable cost is based on specification effort (which is based on factors like complexity of functionalities, number of modules, software-hardware interfaces and new features, network topology) and system validation effort (which is based on factors like protocol type, number of interfaces, number of interrupt routines, software design methodology etc); the efforts may be computed from historical trends.

The external variable cost for hardware design is modeled as the sum of control module design cost, validation and verification cost and wiring design costs. The control modules design cost, is the sum of design cost of each individual module and includes the cost of designing/ redesigning the sub-components and the cost of packaging.

Part Fabrication Cost

The part fabrication cost accounts for the cost of manufacturing the hardware modules. The current supply chain for hardware modules relies almost exclusively on suppliers for fabrication; thus the model includes only the cost of external part fabrication. The fixed costs of production (e.g. investment for tooling) are included as a part of the variable cost. The major costs in this category includes the fabrication cost of individual hardware modules and cost of the interconnects.

The part fabrication cost for hardware module consists of tooling cost (accounted as a share per piece of tooling investment) and piece cost. The piece cost includes the cost of components in the module, and the cost of packaging and interconnecting the modules. Packaging cost accounts for circuit board, housing, wiring and environment factors. Component cost is the sum of the cost of individual components (e.g. processing unit) of a hardware module. The cost of a component can be derived from the cost of the sub-components in the component, e.g. a processing unit may consist of an MPC555 processors, internal and external memory, network interface etc.

The cost of the interconnects is computed from the part cost for harnessing components (e.g. cut leads, connectors, special contacts etc) and part cost for BEC components (relays, circuit breakers, fuses etc.).

Assembly Cost

The *assembly cost* is the cost of integrating the ECS modules into the vehicle during production and accounts for the cost associated with placing and connecting the modules into the vehicle body; the connection cost for supplier parts and sub-systems is accounted for in part fabrication. The internal fixed cost factors include plant space, tools and equipment which would be identical for ECS architecture instances with the same functionality. The internal variable-cost accounts for placement, harnessing, part maintenance, flash and end-of-line (EOL) verification.

Placement cost includes the cost of inserting the system into the vehicle, including fastening, attachment and labor to perform the operations. The harnessing cost is computed from the required number of harnesses and average cost of harness placement. EOL verification is driven by the effort to check the functionality after assembly. The part maintenance cost accounts for the engineering and production effort to keep track of part numbers of new hardware modules and calibration sets for new software modules. The flash cost is computed from the flash time required by all the control modules and flashing cost per unit time.

In-Service Cost

The in-service cost is the usage costs resulting from repair and/or maintenance. Instead of an explicit model, the importance of in-service cost has been qualitatively related to architectural attributes through scenario specific examples [1]. The modeling gap may motivate further research into quantitative links between architecture design attributes and quality, reliability and maintenance costs.

DESIGN CAPTURE OF PRODUCT-LINE ARCHITECTURE

A product-line is “a group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission area” [6]. The product-line architecture consists of a set of architecture instances, one for each product of the product-line. Thus a product-line architecture consists of a description of components and their relationships for each product; the inherent guidelines about the use of the components capture the means for handling required variability among the different products. In our context, a product-line is defined based on the *electronic control system* (ECS) features of a more general product-line, which may possibly include other features like shape, size, and mileage. An ECS product-line architecture is a set of ECS architecture instances where an ECS architecture instance is an instantiation of the ECS product-line architecture for a given product (or *package*, a more conventional term in the automotive domain). To avoid repetitiveness, hereafter we will not use the term ECS .

Formally, a *product-line* PL is a pair $(fset, pset)$ where $fset$ is a set of features and $pset$ is a set of packages. A feature $f \in fset$ is a characterization of the managed features in terms of a set of attributes $Att_f = \{a_1, \dots, a_{n_f}\}$ that are relevant for architecture design. For example, a feature can be a door lock and the subsequent design attributes are given in terms of lines of code, inputs, outputs, execution time, and required fault tolerance. A package p is a pair $(fsubset, v)$ where $fsubset \subseteq fset$ is a subset of features (of the product-line) and v , a positive integer, is the intended volume for that package. Given a product-line PL , a *product-line architecture* $A(PL)$ is a set of architecture instances, one for each package of the product-line. For package $p \in pset$, the architecture instance in product-line architecture PL is $A(PL, p)$. In other words, $A(PL, p)$ implements all the design attributes Att_{f_i} for all features $f_i \in fsubset(p)$. For a product-line PL there may be several product-line architectures. If there are m architecture alternatives, then j -th alternative is denoted as $A_j(PL)$ (where $1 \leq j \leq m$). The architecture instance (of the j -th alternative) is denoted as $A_j(PL, p)$.

alternative) that implements the package p_i of the product-line (let there be n packages in the product-line and $1 \leq i \leq n$), is $A_j(PL, p_i)$. In this paper, we assume that the architecture instance $A_j(PL, p_i)$ correctly implements the features in the package p_i . The

verification problem is challenging and interesting but is outside the scope of the paper. In the next section we will present a formal definition of an architecture instance. The definition used in this work is simpler than the previous model [1]; however it is expressive enough to show the use of the product-line cost model.

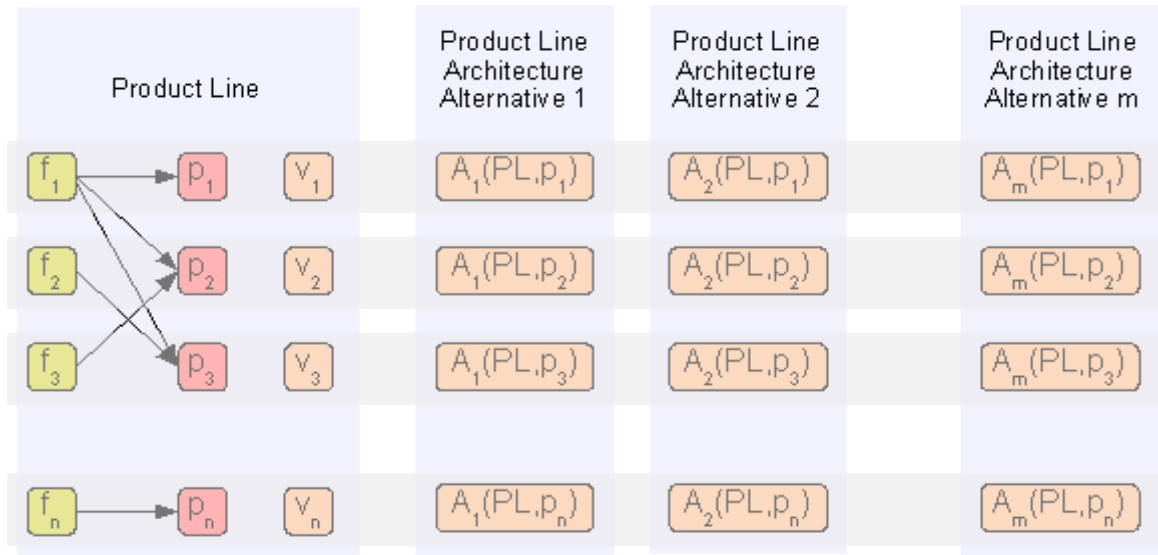


Figure 2: Product-line and Product-line Architecture Alternatives

An *architecture instance* $A_j(PL, p_i)$ is a tuple $(sset, hset, nol, ftime)$ where $sset$ is a set of software modules, $hset$ is a set of hardware modules, nol is number of cut-leads, and $ftime$ is the flash time.

Software Description

A software module is a tuple $(smdl, fpts, kloc, hcomplexity, new)$ where $smdl$ is module name, $fpts$ is the number of function points, $kloc$ is the kilo lines of code per function point for the software module, $hcomplexity$ is a set of complexity factors for designing the software module, and new is the newness of the module. The set of factors $hcomplexity = \{STOR, TIME, VIRT, TURN\}$ are as follows: memory constraint ($STOR$), timing constraint ($TIME$), virtual machine volatility ($VIRT$) and turnaround time ($TURN$). Value of each of the above factors belong to a set $\{NM, HI, VH\}$ where NM , HI and VH denotes “normal”, “high”, and “very high” for the complexity of the software designed. The set of factors are the “computer” attributes used in COCOMO [7]. Other attributes like product, project and personnel effects are assumed to be identical for all modules across all product-line alternatives. Newness factor $new \in \{0,1\}$ denotes whether a software module $smdl$ needs to be tracked for part maintenance ($new=1$) or

not ($new=0$). For architecture instance $A_j(PL, p_i)$, the set of software module names is $sdset(A_j(PL, p_i)) = \{smdl \mid (smdl, \dots) \in sset\}$.

A software module $smdl$ is unique, i.e., if two software modules have identical names, then they have identical function points, kilo lines of code per function point, complexity factors and newness. If a functional module is targeted for two hosts, then they have different software module name. For example, development effort for software modules to operate Auto Door Lock, Power Windows and/or Wind Screen Wiper for MPC 555 vs. MPC 5565 would be different. However development effort for Auto Door Lock on MPC 555 would be identical for any two architecture instances.

A software module is either available off-the-shelf or has to be designed from scratch. If a software module is reused from earlier development phases, the development cost is zero. Software modules are not modified: they are either used or they are not.

Hardware Description

A hardware module is a tuple $(hmdl, num, espec, evalid, cset, pcmpl, new)$ where $hmdl$ is module name, num is the number of instances of the hardware module, $espec$ is specification effort, $evalid$ is validation effort, $cset$ is a set of components, $pcmpl$ is