Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

Improving the accuracy of COPLIMO to estimate the payoff of a software product line

Ruben Heradio*[1], David Fernandez-Amoros†[1], Luis de la Torre‡[1], and Alberto Perez Garcia-Plaza§[1]

[1]ETS de Ingenieria Informatica, Universidad Nacional de Educacion a Distancia, Madrid, Spain

Abstract

Software product line engineering pursues the efficient development of families of similar products. COPLIMO is an economic model that relies on COCOMO II to estimate the benefits of adopting a product line approach compared to developing the products one by one. Although COPLIMO is an ideal economic model to support decision making on the incremental development of a product line, it makes some simplifying assumptions that may produce high distortions in the estimates (e.g., COPLIMO takes for granted that all the products have the same size). This paper proposes a COPLIMO reformulation that avoids such assumptions and, consequently, improves the accuracy of the estimates. To support our proposal, we present an algorithm that infers the additional information that our COPLIMO reformulation requires from feature diagrams, which is a widespread notation to model the domain of a product line.

1 Introduction

Product Line (PL) engineering enables the transition from handcrafting one-of-a-kind solutions towards automated manufacturing of complete portfolios of software products. The fundamental idea of the approach is to undertake the development of a set of products as a single, coherent development task. Products are built from a Core Asset Base (CAB), a collection of artifacts that have been designed specifically for use across the portfolio [1].

Many managers favor an incremental approach to PL adoption, one that first tackles areas of highest and most readily available commonality, earning payback early in the adoption cycle [2]. Under this approach, the organization plans from the beginning to develop a PL. It develops part of the CAB, including the architecture and some of the assets, and then develops one or more products. In the next increment, it develops a portion of the rest of the CAB and develops additional products. Over time, it evolves more of the CAB in parallel with new product development. The main decisions that can be made in each increment are:

1. Adding assets to the CAB to increase the portion of the products that is built by combining core assets.

*rheradio@issi.uned.es
†david@lsi.uned.es
‡ldelatorre@bec.uned.es
§alpgarcia@lsi.uned.es

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

2. Removing from the CAB those assets that are never or rarely used, causing unnecessary complexity and maintenance costs.

3. Making the core assets easier to reuse (e.g., converting a white–box reused asset into a black–box reused one[1]). Such decision usually implies to invest some effort in making the assets more reusable. On the other hand, the investment is recovered every time the assets are used to build the products.

This paper argues that the COnstructive Product Line Investment MOdel (COPLIMO) [4, 5] is an ideal economic model to support decision making on the incremental development of PLs. However, COPLIMO makes some simplifying assumptions that may produce high distortions in the estimates (e.g., COPLIMO takes for granted that all the products have the same size, all the core assets are reused by all the products...). This work identifies such assumptions and proposes the modification of COPLIMO to avoid them, improving the accuracy of the estimates.

Our improvement requires to know how many products in a PL implement a feature f (i.e., $\#\mathcal{P}_f$). In software product line engineering, Feature Diagrams (FDs) are a popular means to represent the similarities and differences within a family of related systems [6]. In this paper, we propose an algorithm to infer $\#\mathcal{P}_f$ from FDs. Our algorithm works for large diagrams since its time complexity is just quadratic on the number of features included in the FD.

The remainder of the paper is structured as follows: section 2 outlines available economic models for PLs and justifies the suitability of COPLIMO to assess domain engineers during the incremental development of a PL. The section also summarizes the state of art on computing $\#\mathcal{P}_f$ for a FD. Section 3 presents the terminology about PLs and FDs that will be used throughout the paper. Section 4 summarizes COPLIMO and identifies its limitations. Section 5 presents our proposal to improve the accuracy of COPLIMO. Section 6 presents a case of study that illustrates the benefits of our approach. Section 7 describes our algorithm to compute $\#\mathcal{P}_f$. Finally, section 8 presents the conclusions.

2 Related work

The following sections survey existing economic models for PLs, different proposals for FDs and the state of the art of product counting for FDs.

2.1 Economic models for product lines

There has been a great deal of research on the economics of software reuse, and a large number of economic models have been proposed. For instance, Mili et al. [7] reference 40 different models and analyze 17 of them. However, general models in the context of software reuse can only be applied in a restricted way, since PL development involves some fundamental assumptions that are not reflected in these models [8]. To overcome this situation, several economic models oriented specifically to PLs have been proposed, which can be classified in the following categories:

1. *Primitive models* that are not based on any other PL model. The following proposals fall in this category:

   (a) SIMPLE [8] enumerates a set of cost functions that can be used to construct equations to answer a number of questions, such as whether the PL approach is the best option for development and what the Return on Investment (ROI) is for this approach. Unfortunately, SIMPLE is just an abstract framework that does not provide any implementation for cost functions.

[1] An asset is *black-box reused* if it is not necessary to understand the asset implementation to reuse it. Otherwise, the asset is *white-box reused* [3].

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

(b) Poulin's model [9] distinguishes between the cost of reuse an asset (the Relative Cost of Reuse, RCR) and the cost of making it reusable (the Relative Cost of Writing for Reuse, RCWR). So, the model may estimate the payoffs of making the core assets more reusable. RCR and RCWCR are defined in relative terms to the cost required to develop software for one-time use. For instance, RCR is the proportion of the effort that it takes to reuse software compared to the cost normally incurred to develop it for one-time use (e.g., a feature has RCR = 0.2 if it can be reused for only 20% of the cost of implementing it). A major drawback of Poulin's model is that it only considers software size to estimate the cost of building software for one-time use. As noted in [10] and [11], size is not enough to estimate effort for software development in general, because additional information has to be considered, such as the expertise of the development team, projected technologies to be used, software operational requirements (e.g., reliability, performance, maintainability) and so forth.

(c) Peterson's model [12] characterizes the PL savings estimating the redundant efforts needed to develop independently the products compared to the efforts consumed by the PL approach. Unfortunately, Peterson's model also only takes into consideration software size to estimate the effort of building the products independently.

(d) COPLIMO, as Poulin's model, distinguishes between RCR and RCWR. What makes COPLIMO stand out is relying on COCOMO II [10] to estimate software development effort. Thus, COPLIMO enriches software size with organizational information, a number of scale factors and effort multipliers.

2. *Derived models* that enrich the primitive ones. For instance:

(a) Nobrega et al. [13] extend SIMPLE with a viewpoint layer that provides different economic perspectives to the PL stakeholders (e.g., corporate managers, domain engineering teams, application engineering teams, individual producers of reusable parts...).

(b) Lamine et al. [14] extend Poulin's model with a viewpoint layer as well.

(c) Critical aspects to the accuracy and usefulness of an estimate are often unavailable or inaccurate (e.g., the correct size of software, historical data...). Ganesan et al. [15] propose the usage of Monte-Carlo simulation in combination with SIMPLE to manage uncertain variables for economic models.

(d) Chen et al. [16] enrich COPLIMO with Discrete Event System Specification to simulate the effects of different PL adoption approaches on the PL evolution.

(e) qCOPLIMO [5] is derived from COPLIMO and COQUALMO [10]. It models the effect on the PL savings of software quality costs, which are spent on removing undetected defects after product release.

Since COPLIMO (i) relies on COCOMO II, providing complex cost functions, and (ii) distinguishes between RCR and RCWR, supporting the assessment of making the core assets more reusable, it is an ideal economic model to support decision making for the incremental development of PLs. Due to the improvement of the accuracy of a primitive model implies the improvement of its derived models, our work not only improves COPLIMO, but also the proposals derived from it.

2.2 Feature Diagrams

The domain of a PL should be carefully scoped, identifying the common and variable features of its products and the interdependencies between features. In ill-scoped domains, relevant features may not be

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

implemented, and implemented features may never be used, causing unnecessary complexity and both development and maintenance costs [17]. To avoid these serious problems, PL domains are usually modeled by means of FDs. This paper proposes how to use the information modeled with FD to improve the accuracy of COPLIMO estimates.

Since the first language was proposed by the Feature–Oriented Domain Analysis (FODA) methodology in 1990 [18], a number of extensions and alternative languages have been devised to model variability in families of related systems:

1. As part of the following methods: FORM [19], FeatureRSEB [20], Generative Programming [17], PLUSS [21].

2. In the work of the following authors: Riebisch et al. [22], van Gurp et al. [23], van Deursen et al. [24], Gomaa [25], Pohl [26].

3. As part of the following tools: Gears [27] and pure::variants [28].

The equivalences among most of the available notations for FDs have been studied by Schobbens et al. [6, 29]. In this paper, we will use the notation proposed by Czarnecki et al. [17].

2.3 Computing the number of products that implement each feature

The COPLIMO improvement we propose in this paper requires to know how many products implement a feature. Since FDs implicitly model such information, it can be inferred from them.

Analyzing FDs is an error–prone and tedious task, and it is unfeasible to do manually with large–scale feature diagrams. For this reason, the automated analysis of feature diagrams is an active area of research in the PL community [30].

Existing proposals for computing the number of products that implement each feature translate FDs into propositional logic formulas, which are processed by off-the-shelf tools. Benavides et al. [31, 32] translate feature diagrams into propositional logic formulas (i.e., FD $\rightsquigarrow \psi$). Off-the-shelf tools, such as Boolean Satisfiability (SAT) solvers, Constraint Satisfaction Problem (CSP) solvers and Binary Decision Diagrams (BDD), are then used to enumerate all the different sets of variable assignments that satisfy the logic formula ψ. Each one of these sets represents a particular product. There is a correspondence between features and variables, so that if a feature f is encoded in ψ by a boolean variable v, then f is included in the product represented by a set s if the value of v in s is true. Hence, commonality of f is calculated by counting the number $\#\mathcal{P}_f$ of sets where v is true.

As noted by Sang et al. [33], any backtracking SAT algorithm can be trivially extended to one that counts the number of satisfying assignments by simply forcing it to backtrack whenever a solution is found. However, such a simple approach, is unfeasible for all but the smallest problem instances. As Benavides reports [32], CSP technology scales even worse than SAT-solvers to compute the number of satisfying assignments. The approach can be improved by using tools specifically designed for counting the number of valid assignments that a formula has, such as *relsat* [34] and *cachet* [33, 35], two state of the art model counters for propositional logic.

However, the worst–case running of all the cited proposals run is exponential (i.e., the computation may be unfeasible for large diagrams). To overcome such situation, in section 7 we propose an algorithm that computes the number of products that implement a feature in just quadratic time.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

3 An example of feature diagram

This section introduces the terminology that we will use throughout the paper by means of a simplified version of a PL example proposed in [17]. The aim of the example is to develop a portfolio of list containers. The PL scope is represented in Figure 1 by a FD, which is a hierarchically arranged set of features with different relations among them. Figure 1 includes three kinds of relations[2]: mandatory (pointed by simple edges ending with a filled circle; i.e., all Lists in the portfolio have an Ownership feature), alternative (pointed by edges connected by an arc; i.e., External reference, Owned reference and Copy are the mutually exclusive values for Ownership) and optional (pointed by simple edges ending with an empty circle; i.e., a List may have the Tracing feature). In this particular example, the meanings of the features are:

1. Ownership specifies how a list stores its elements:
   (a) External reference: the list keeps references to the original elements and is no responsible for element deallocation.
   (b) Owned reference: the list keeps references and is responsible for element deallocation.
   (c) Copy: the list keeps copies of the original elements and is responsible for their allocation and deallocation.
2. Tracing indicates if a list traces its operation by logging function calls to the console.

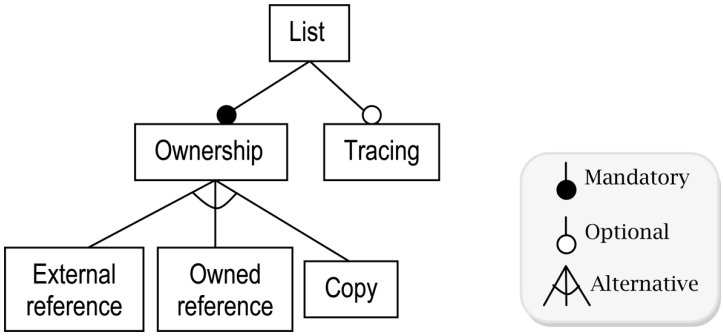


Figure 1: Feature diagram for a portfolio of list containers

In our lexicon, $\mathcal{A}$ denotes the set of assets necessary to make all the products of the portfolio. Let us assume that the FD terminal features[3] External reference, Owned reference, Copy and Tracing are respectively implemented by the assets External reference, Owned reference, Copy and Tracing. Thus, $\mathcal{A}$ is given by Equation 1.

$$\mathcal{A} = \{\text{External reference, Owned reference, Copy, Tracing}\} \quad (1)$$

Most PLs are developed iteratively, adding new assets to the CAB to increase the PL degree of reuse in successive development cycles. Thus, CAB $\subseteq \mathcal{A}$. To simplify the example, we will suppose all assets belong to the CAB (i.e., CAB $= \mathcal{A}$).

[2] In addition to *mandatory*, *optional* and *alternative* relations, notations for FDs usually consider *or* relations as well. A feature f with *or* subfeatures means that, whenever f is implemented by a product, none, one or more of the f subfeatures can be implemented by the product [36].

[3] FD intermediate nodes (e.g., List and Ownership) are typically used just for decomposition purposes [6], i.e., each product corresponds to a different configuration of terminal features.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

A product p is denoted by the set of assets that make it (e.g., the product p = {External reference, Tracing} is composed of the assets External reference and Tracing).

$\mathcal{P}$ denotes the set of products included in the portfolio. In this example, $\mathcal{P}$ is given by Equation 2, which excludes products specified by invalid feature combinations (e.g., {External reference, Copy} $\notin \mathcal{P}$ because External reference and Copy are alternative features and consequently cannot be part of the same product). Furthermore, not all valid products modeled by a FD are necessarily of economic interest. Hence, $\mathcal{P}$ might also exclude the uninteresting products.

$$\begin{aligned}\mathcal{P} = \{ & \{\text{External reference}\}, \{\text{Owned reference}\}, \{\text{Copy}\} \\ & \{\text{External reference, Tracing}\}, \{\text{Owned reference, Tracing}\}, \\ & \{\text{Copy, Tracing}\} \}\end{aligned} \tag{2}$$

We denote the cardinal of a set S by $\#S$ (e.g., $\#\mathcal{A} = 4$ and $\#\mathcal{P} = 6$ because $\mathcal{A}$ and $\mathcal{P}$ have 4 and 6 elements, respectively).

$\mathcal{P}_a$ is the subset of $\mathcal{P}$ composed of the products that include the asset a (e.g., $\mathcal{P}_{\text{Copy}} = \{\{\text{Copy}\}, \{\text{Copy, Tracing}\}\}$).

4 COPLIMO

COPLIMO estimates the payoff of a PL by analogy. As depicted in Figure 2, COPLIMO starts estimating the development costs of a particular product p_1. Then, supposing that all products in the scope of a PL are quite similar (i.e., $p_1 \approx p_2 \approx ... \approx p_n$), it extrapolates the costs of p_1 to compare the costs of building all the products under a PL approach and building them one by one.

Sections 4.1 and 4.2 outline how COPLIMO estimates the costs of developing a particular product and a family of products, respectively.

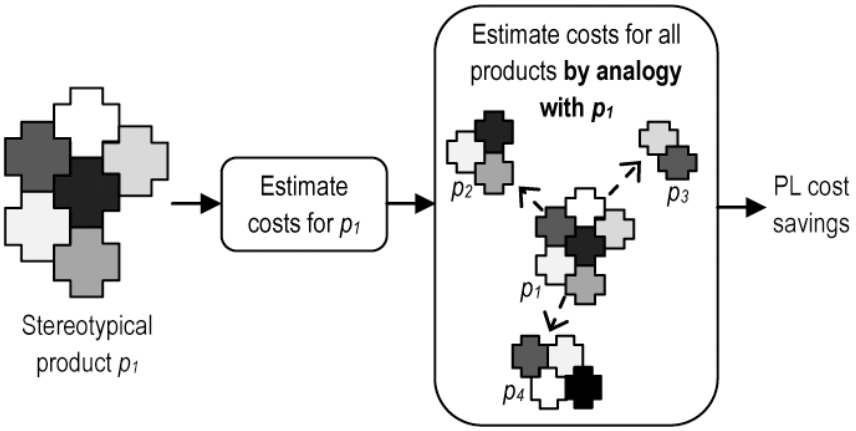


Figure 2: COPLIMO approach to estimate the cost savings of a PL

4.1 Estimating the costs for a particular product

COPLIMO relies on COCOMO, which uses Equation 3 to estimate the amount of effort in *Person-Months* (PM) it will take to develop a software product; where:

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

- A is an organization-dependent constant that approximates the organization productivity.
- Size is an estimation of the software size. COCOMO uses 3 metrics to measure the size: (thousands of) *Source Lines Of Code* (KSLOC), Function Points and Object Points.
- B is an aggregation of scale factors. It accounts for the relative economies or diseconomies of scale. If $B < 1$, there are economies of scale (i.e., the productivity increases as the product size is increased). On the other hand, if $B > 1$, there are diseconomies of scale (two main factors for diseconomies are the growth of interpersonal communications overhead and the growth of large-system integration overhead).
- EM_i are effort multipliers used to adjust the PM effort to reflect the product under development. An example of EM_i is SCED, which stands for *required development schedule* and adjusts the PM effort according to a percentage of schedule stretch-out or acceleration; i.e., accelerated schedules tend to require more effort in the earlier phases of product development (to eliminate risks and refine the architecture) and in the later phases (to accomplish more testing and documentation in parallel).

$$\text{PM} = A \times \text{Size}^B \times \prod_i \text{EM}_i \tag{3}$$

4.2 Estimating the costs for the whole PL

COPLIMO estimates the *Product Line Savings* (PLS) and the PL *Return Of Investment* (ROI)[4] with Equations 4 and 5. $\text{PMR}(N)$ and $\text{PMNR}(N)$ stand for the PM effort to develop N products with reuse and without it (i.e., PL vs individual development).

$$\text{PLS}(N) = \text{PMNR}(N) - \text{PMR}(N) \tag{4}$$

$$\text{ROI}(N) = \frac{\text{PLS}(N)}{|\text{PLS}(1)|} \tag{5}$$

According to COCOMO, factors A, B and SCED have to be adjusted at project level. Nevertheless, Size and the rest of EM_i apply to individual project components. Surprisingly, COPLIMO obviates such differentiation and manages all cost drivers at project level. Furthermore, COPLIMO assumes that all products have the same size. Hence, $\text{PMNR}(N)$ is calculated by Equation 6.

$$\text{PMNR}(N) = N \times A \times \text{Size}^B \times \prod_i \text{EM}_i \tag{6}$$

COPLIMO also makes the following simplifying assumptions:

1. All core assets are reused by all products; i.e., $\forall p \in \mathcal{P} \cdot \text{CAB} \subseteq p$. Hence, the CAB development cost is approximated by $|\text{PLS}(1)|$
2. COPLIMO uses acronyms PFRAC, RFRAC and AFRAC to denote the proportions of a product that are unique, composed of black-box reused assets and composed of white-box reused assets, respectively. COPLIMO assumes that all products have the same PFRAC, RFRAC and AFRAC; i.e., $\forall p, p' \in \mathcal{P} \cdot (\text{PFRAC}(p) = \text{PFRAC}(p')) \wedge (\text{RFRAC}(p) = \text{RFRAC}(p')) \wedge (\text{AFRAC}(p) = \text{AFRAC}(p'))$

[4] The PL approach is not always the best economic choice for developing a family of related products. Since PLS may be negative, the absolute value of PLS(1) is used in Equation 5.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

As a result, PMR(N) is calculated by Equation 7, where:

- RCWR stands for Relative Cost of Writing for Reuse. It represents the effort to make software reusable across the PL. RCWR is calculated by Equation 8, where RUSE (Development for Reuse) addresses the added costs of domain engineering, PL architecture developments and of *generalizing* the code to cover a range of applications; DOCU (Degree of Documentation) is used to capture the added costs of documenting reusable software for family-wide use; and RELY (Required software Reliability) is used to capture the added costs of verifying and validating that the reusable software will work successfully across the PL.
- Regarding the cost of reusing the CAB, COPLIMO distinguishes between black-box and white-box reused assets. Black-box reuse cost is set by the Assessment and Assimilation (AA) factor. White-box reuse cost is set by the Adaptation Adjustment Modifier (AAM). AAM is calculated by Equations 9 and 10, where AAF Adaptation Adjustment Factor measures the amount of DM (percentage of Design Modified), CM (percentage of Code Modified) and IM (percentage of Integration effort for integrating the adapted software). Finally, SU (Software Understanding) measures the understandability of the core assets and UNFM (Programmer Unfamiliarity) measures the programmers' relative unfamiliarity with the core assets.

$$\mathrm{PMR}(N) = \begin{cases} \mathrm{PMNR}(1)\times \\ \quad \times \Big(\mathrm{PFRAC}+\mathrm{RCWR}\times(\mathrm{RFRAC}+\mathrm{AFRAC})\Big) & \text{if } N=1 \\ \mathrm{PMR}(1)+(N\text{-}1)\times \mathrm{PMNR}(1)\times \\ \quad \times(\mathrm{PFRAC}+\mathrm{RFRAC}\times\frac{\mathrm{AA}}{100}+\mathrm{AFRAC}\times\mathrm{AAM}) & \text{if } N>1 \end{cases} \tag{7}$$

$$\mathrm{RCWR} = \mathrm{RUSE}\times\mathrm{DOCU}\times\mathrm{RELY} \tag{8}$$

$$\mathrm{AAM} = \begin{cases} \frac{\mathrm{AA}+\mathrm{AAF}\left(1+(0.02\times\mathrm{SU}\times\mathrm{UNFM})\right)}{100} & \text{if } \mathrm{AAF}\le 50 \\ \frac{\mathrm{AA}+\mathrm{AAF}+(\mathrm{SU}\times\mathrm{UNFM})}{100} & \text{if } \mathrm{AAF}>50 \end{cases} \tag{9}$$

$$\mathrm{AAF} = (0.4\times\mathrm{DM})+(0.3\times\mathrm{CM})+(0.3\times\mathrm{IM}) \tag{10}$$

5 Improving the accuracy of COPLIMO estimates

In order to estimate the payoff of a PL, COPLIMO supposes an extreme homogeneity among the products. In particular, it makes the following simplifying assumptions:

- A_1: all the products have the same size.
- A_2: all the core assets are reused by all the products.
- A_3: all the products have the same proportions of unique assets, black-box reused assets and white-box reused assets.
- A_4: COCOMO parameters Size and all EM_i are estimated at project level, instead of being estimated for each asset.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

As we will exemplify in Section 6, such assumptions may produce problematic distortions in the estimates. To overcome that problem, in this section we reformulate COPLIMO. Figure 3 outlines our approach, that includes the following steps:

1. First of all, the costs are estimated for each asset, so assumption A_4 is avoided. This step is described in Section 5.1.
2. Then, the costs of each product are calculated by adding the costs of the assets that it includes. As we will see in Section 4.2, in order to avoid assumptions A_1, A_2 and A_3, it is necessary to know $\#\mathcal{P}_a$. In section 7, we propose how to compute $\#\mathcal{P}_a$ from a FD.

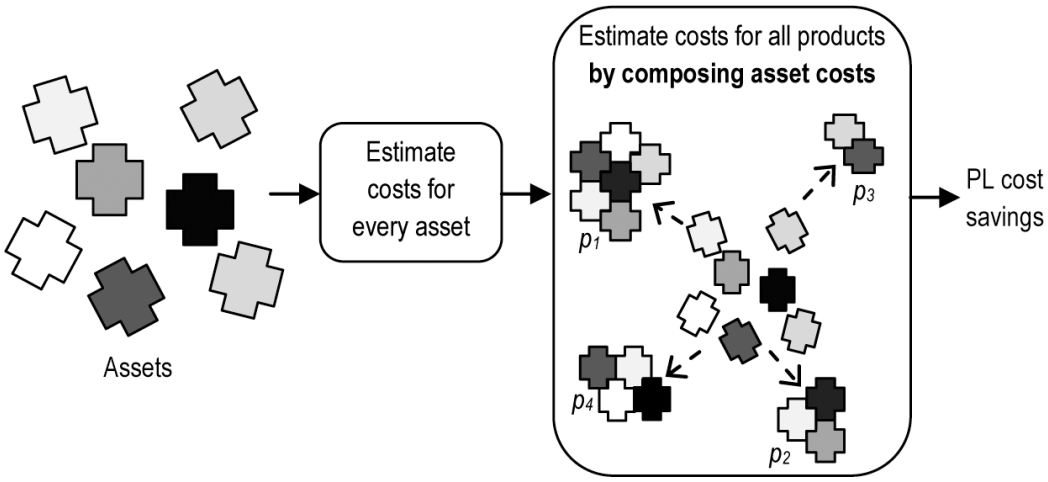


Figure 3: Our approach to estimate the cost savings of a PL

5.1 Estimating the costs for an asset

According to pages 55-56 in [10], estimating effort for systems comprised of multiple components by adding up the estimation for each component ignores effort due to integration of the components. To get over this problem, COCOMO proposes an alternative method that we have adapted for PLs as follows:

1. Sum the sizes for all of the assets, Size_i, to yield the aggregate size of an average product:

$$\text{Size}_{\text{Aggregate}}(\bar{p}) = \frac{\sum_{a \in \mathcal{A}} (\text{Size}(a) \times \#\mathcal{P}_a)}{\#\mathcal{P}} \tag{11}$$

2. Apply the project-level drivers, the Scale Factors and the SCED Cost Driver, to the aggregated size to derive the overall basic effort for a product,PM_{Basic}:

$$\text{PM}_{\text{Basic}}(\bar{p}) = A \times (\text{Size}_{\text{Aggregate}}(\bar{p}))^B \times \text{SCED} \tag{12}$$

3. Determine each asset's basic effort, $\text{PM}_{\text{Basic}_i}$, by apportioning the overall basic effort to each asset based on its contribution to the aggregate size:

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

$$\mathrm{PM}_{\mathrm{Basic}}(a) = \mathrm{PM}_{\mathrm{Basic}}(\bar{p}) \times \frac{\mathrm{Size}(a)}{\mathrm{Size}_{\mathrm{Aggregate}}(\bar{p})} \tag{13}$$

4. Apply the asset-level cost drivers (excluding SCED) to each asset's basic effort:

$$\mathrm{PM}(a) = \mathrm{PM}_{\mathrm{Basic}}(a) \times \prod_j \mathrm{EM}_j \tag{14}$$

5.2 Estimating the costs for the whole PL

PLS and ROI are estimated with Equations 15-21, where CCAB stands for the Cost of developing the CAB.

$$\mathrm{PLS}(\mathcal{P}) = \mathrm{PMNR}(\mathcal{P}) - \mathrm{PMR}(\mathcal{P}) \tag{15}$$

$$\mathrm{ROI}(\mathcal{P}) = \frac{\mathrm{PLS}(\mathcal{P})}{\mathrm{CCAB}(\mathcal{P})} \tag{16}$$

PMNR is estimated with Equation 17. Compared to the original Equation 6, it does not assume that all products have the same size (assumption A_1). The new equation estimates the costs of a product by adding up the costs of its assets.

$$\mathrm{PMNR}(\mathcal{P}) = \sum_{a \in \mathcal{A}} \left(\mathrm{PM}(a) \times \#\mathcal{P}_a\right) \tag{17}$$

$$\mathrm{PMR}(\mathcal{P}) = \mathrm{CCAB}(\mathcal{P}) + \mathrm{CUNIQ}(\mathcal{P}) + \mathrm{CRCAB}(\mathcal{P}) \tag{18}$$

$$\mathrm{CCAB} = \sum_{a \in \mathrm{CAB}} \left(\mathrm{PM}(a) \times \mathrm{RCWR}(a)\right) \tag{19}$$

$$\mathrm{CUNIQ}(\mathcal{P}) = \sum_{a \notin \mathrm{CAB}} \mathrm{PM}(a) \tag{20}$$

$$\mathrm{CRCAB}(\mathcal{P}) = \sum_{a \in \mathrm{CAB}} \left(\mathrm{PM}(a) \times \#\mathcal{P}_a \times \begin{cases} \frac{\mathrm{AA(a)}}{100} & \text{if } a \text{ is BBR} \\ \mathrm{AAM(a)} & \text{if } a \text{ is WBR} \end{cases} \right) \tag{21}$$

6 A case of study

Section 6.1 extends the example introduced in section 3 completing the input to COPLIMO and our improved model. Afterwards, ROI and CSAV are estimated in sections 6.2 and 6.3 by using the original COPLIMO and our COPLIMO improvement, respectively. Finally, section 6.4 summarizes the estimation errors caused by COPLIMO simplifying assumptions and how our model overcomes such errors.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

6.1 Input values to the economic models

In the example, we will make the following assumptions:

1. All assets belong to the CAB and are black-box reused.
2. In order to keep the calculations as simple as possible, we will suppose the following values for the parameters A, B and $\prod_i \mathrm{EM}_i$ of COCOMO:
   (a) $A = 2.94$; i.e., A has the standard value proposed in [10] after calibrating the COCOMO model for 161 projects.
   (b) B is 1; i.e., the economies and diseconomies of scale are in balance.
   (c) the value of all effort multipliers is 1, which is the value proposed in [10] for an average project. Thus, $\prod_i \mathrm{EM}_i = 1$.
3. CORG=0; i.e., the adoption of the PL approach does not imply organizational costs.

Table 1 summarizes the rest of the input values that will be used in the example. For instance, the asset External reference appears in 2 products ($\#\mathcal{P}_a$=2), has a size of 0.4 KSLOC and proportions of added costs for reuse RUSE[5]=1.15, for documenting software for family-wide use DOCU[6]=1 and for verifying and validating RELY=1. So, it has a relative cost of writing for reuse of 1.15 (i.e., RCWR = RUSE × DOCU × RELY). Finally, it has a cost factor of assessment and assimilation AA[7]=2.

Asset a	$\#\mathcal{P}_a$	**Size**(a)	**RUSE**(a)	**DOCU**(a)	**RELY**(a)	**RCWR**(a)	**AA**(a)
External reference	2	0.4	1.15	1	1	1.15	2
Owned reference	2	0.5	1.15	1	1.1	1.26	2
Copy	2	0.6	1.15	1	1.1	1.26	2
Tracing	3	0.6	1.15	1	1	1.15	2

Table 1: Input values to the economic models

6.2 Original COPLIMO estimates

Table 2 summarizes the different results that COPLIMO produces according to the stereotypical product p chosen to make the calculations[8]. Equations 22-28 describe the calculations for the case p = {External reference}.

$$\mathrm{CNPL}(\#\mathcal{P}) = \#\mathcal{P} \times A \times \mathrm{Size}(p)^B \times \prod_i \mathrm{EM}_i(p) =$$
$$= 6 \times 2.94 \times 0.4^1 \times 1 = 7.05 \quad (22)$$

$$\mathrm{CNPL}(1) = 1 \times 2.94 \times 0.4^1 \times 1 = 1.17 \quad (23)$$

[5]Due to all assets belong to the CAB, their RUSE=1.15, which means a *very high* rating level of reuse. [10] provides tables to convert the linguistic rating levels of RUSE, DOCU, RELY and AA (e.g., *low*, *nominal*...) into their numerical equivalence (e.g., 0.95, 1...).

[6]DOCU=1 and RELY=1 expresses a *nominal* rating level.

[7]AA=2 indicates that the assessment and assimilation effort is limited to basic module search and documentation.

[8]Numbers are truncated to 2 digits.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

Selected product p	CNPL($\#\mathcal{P}$)	CPL($\#\mathcal{P}$)	CCAB	CSAV($\#\mathcal{P}$)	ROI($\#\mathcal{P}$)
{External reference}	7.05	1.47	0.17	5.58	31.66
{Owned reference}	8.82	2	0.38	6.81	17.49
{Copy}	10.58	2.4	0.46	8.17	17.49
{External reference, Tracing}	17.64	3.67	0.44	13.96	31.66
{Owned reference, Tracing}	19.4	4.22	0.67	15.17	22.61
{Copy, Tracing}	21.16	4.61	0.73	16.55	22.61

Table 2: Original COPLIMO estimates

$$\begin{aligned}\mathrm{CPL}(1) &= \mathrm{CNPL}(1) \times \\ &\times \Big(\mathrm{PFRAC}(p) + \mathrm{RCWR}(p) \times \big(\mathrm{RFRAC}(p) + \mathrm{AFRAC}(p)\big)\Big) = \\ &= 1.17 \times \big(0 + 1.15 \times (1 + 0)\big) = 1.35\end{aligned} \tag{24}$$

$$\begin{aligned}\mathrm{CPL}(\#\mathcal{P}) &= \mathrm{CPL}(1) + (\#\mathcal{P} - 1) \times \mathrm{CNPL}(1) \times \big(\mathrm{PFRAC}(p) + \\ &+ \mathrm{RFRAC}(p) \times \frac{\mathrm{AA}}{100} + \mathrm{AFRAC}(p) \times \mathrm{AAM}\big) = \\ &= 1.35 + (6 - 1) \times 1.17 \times \Big(0 + 1 \times \frac{2}{100} + 0\Big) = 1.47\end{aligned} \tag{25}$$

$$\mathrm{CINV}(\#\mathcal{P}) = \mathrm{CCAB} = |\mathrm{CNPL}(1) - \mathrm{CPL}(1)| = |1.17 - 1.35| = 0.17 \tag{26}$$

$$\mathrm{CSAV}(\#\mathcal{P}) = \mathrm{CNPL}(\#\mathcal{P}) - \mathrm{CPL}(\#\mathcal{P}) = 7.05 - 1.47 = 5.58 \tag{27}$$

$$\mathrm{ROI}(\mathcal{P}) = \frac{\mathrm{CSAV}(\#\mathcal{P})}{\mathrm{CINV}(\#\mathcal{P})} = \frac{5.58}{0.17} = 31.66 \tag{28}$$

6.3 Improved COPLIMO estimates

In contrast to COPLIMO, our improved model produces a unique estimation that is described in section 6.3.2. To make such estimation, the calculation of PM is firstly presented in section 6.3.1.

6.3.1 Calculating PM(a)

Table 3 summarizes the values of PM for all assets. Equations 29-32 describe the calculations for PM(External reference)[9].

$$\begin{aligned}\mathrm{Size}_{\mathrm{Aggregate}}(\bar{p}) &= \frac{\sum_{a \in \mathcal{A}} (\mathrm{Size}(a) \times \#\mathcal{P}_a)}{\#\mathcal{P}} = \\ &= \frac{0.4 \times 2 + 0.5 \times 2 + 0.6 \times 2 + 0.6 \times 3}{6} = 0.8\end{aligned} \tag{29}$$

$$\mathrm{PM}_{\mathrm{Basic}}(\bar{p}) = A \times \big(\mathrm{Size}_{\mathrm{Aggregate}}(\bar{p})\big)^B \times \mathrm{SCED} = 2.94 \times 0.8^1 \times 1 = 2.35 \tag{30}$$

[9]The value of all effort multipliers is 1. So, SCED=1 in Equation 30.

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

Asset a	PM(a)
External reference	1.17
Owned reference	1.47
Copy	1.76
Tracing	1.76

Table 3: PM(a)

$$\begin{aligned}&\mathrm{PM}_{\mathrm{Basic}}(\text{External reference}) = \\ &= \mathrm{PM}_{\mathrm{Basic}}(\bar{p}) \times \frac{\mathrm{Size}(\text{External reference})}{\mathrm{Size}_{\mathrm{Aggregate}}(\bar{p})} = \\ &= 2.35 \times \frac{0.4}{0.8} = 1.17\end{aligned} \tag{31}$$

$$\begin{aligned}&\mathrm{PM}(\text{External reference}) = \\ &= \mathrm{PM}_{\mathrm{Basic}}(\text{External reference}) \times \prod_j \mathrm{EM}_j = 1.17 \times 1 = 1.17\end{aligned} \tag{32}$$

6.3.2 Calculating CSAV and ROI

Table 4 summarizes the estimation produced by our improved model, which is described by Equations 33-39.

	CNPL($\mathcal{P}$)	**CPL($\mathcal{P}$)**	**CCAB**	**CSAV($\mathcal{P}$)**	**ROI($\mathcal{P}$)**
Improved COPLIMO	14.11	7.75	7.47	6.35	0.85

Table 4: Improved COPLIMO estimates

$$\begin{aligned}&\mathrm{CNPL}(\mathcal{P}) = \sum_{a \in \mathcal{A}} \left(\mathrm{PM}(a) \times \#\mathcal{P}_a\right) = \\ &= 1.17 \times 2 + 1.47 \times 2 + 1.76 \times 2 + 1.76 \times 3 = 14.11\end{aligned} \tag{33}$$

$$\mathrm{CRCAB}(\mathcal{P}) = \sum_{a \in \mathrm{CAB}} \left(\mathrm{PM}(a) \times \#\mathcal{P}_a \times \frac{\mathrm{AA}}{100}\right) = 14.11 \times \frac{2}{100} = 0.28 \tag{34}$$

$$\begin{aligned}&\mathrm{CCAB} = \sum_{a \in \mathrm{CAB}} \left(\mathrm{PM}(a) \times \mathrm{RCWR}(a)\right) = \\ &= 1.17 \times 1.15 + 1.47 \times 1.26 + 1.76 \times 1.26 + 1.76 \times 1.15 = 7.47\end{aligned} \tag{35}$$

$$\mathrm{CINV}(\mathcal{P}) = \mathrm{CORG}(\mathcal{P}) + \mathrm{CCAB} = 0 + 7.47 = 7.47 \tag{36}$$

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

$$\text{CPL}(\mathcal{P}) = \text{CINV}(\mathcal{P}) + \text{CUNIQ}(\mathcal{P}) + \text{CRCAB}(\mathcal{P}) = 7.47 + 0 + 0.28 = 7.75 \tag{37}$$

$$\text{CSAV}(\mathcal{P}) = \text{CNPL}(\mathcal{P}) - \text{CPL}(\mathcal{P}) = 14.11 - 7.75 = 6.35 \tag{38}$$

$$\text{ROI}(\mathcal{P}) = \frac{\text{CSAV}(\mathcal{P})}{\text{CINV}(\mathcal{P})} = \frac{6.35}{7.47} = 0.85 \tag{39}$$

6.4 Contrasting the estimations

Figure 4 compares the estimations produced by the original COPLIMO and our improved model. It emphasizes the COPLIMO errors caused by the following assumptions:

1. *Every product reuses all assets included into the CAB.* This assumption does not hold in our example (neither any real scenario). For that reason, CCAB is underestimated with Equation 26. Figure 4.a shows that the CCAB calculated by COPLIMO using any product is much lower than the CCAB calculated by our improved model. The CCAB computed by our model is the addition of the effort of developing the assets included in the CAB (i.e., the result provided by our model is consistent with the CCAB definition: *the Cost of developing the CAB*). On the contrary, COPLIMO estimates a CCAB that is even minor than the cost of developing any of the assets (see columns CCAB and PM(a) in Tables 2 and 3, respectively). This error propagates throughout all COPLIMO calculations:
   (a) underestimating CPL (see Figure 4.b).
   (b) overestimating CSAV and ROI (see Figures 4.c and 4.d).
2. *Products are extremely homogeneous.* Clements et al. [8] define a measure, named *homogeneity*, that characterizes how homogeneous the products are. The metric varies from 0 to 1, where 0 indicates that the products are all totally unique and 1 indicates that there is only one product. Equation 40 expresses homogeneity in our common lexicon and calculates it for the example. Since the example has low homogeneity, COPLIMO simplifying assumptions produce strong variations in the estimations depending on the chosen product. For instance, Figure 4.b shows that COPLIMO estimations of CNPL variate from 7.05 to 21.16. In contrast, our improved model estimates a balanced value of 14.11, that is approximately the average of the CNPL values calculated by COPLIMO.

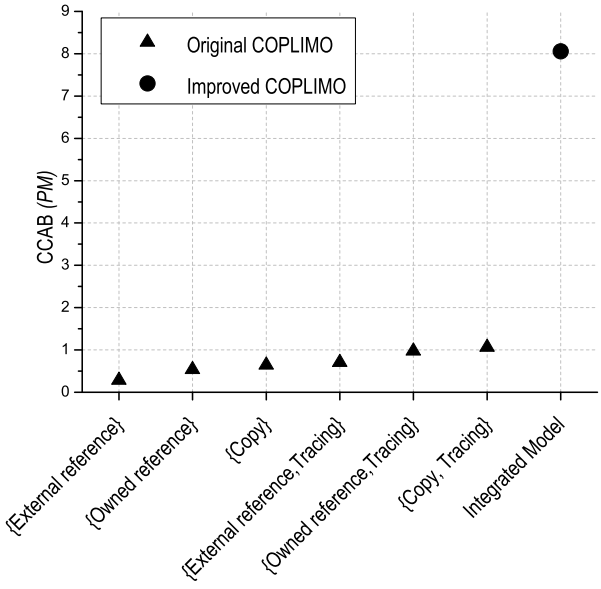
$$\text{Homogeneity}(\mathcal{P}) = \frac{\sum_{a \in \mathcal{A}} \frac{\#\mathcal{P}_a}{\#\mathcal{P}}}{\#\mathcal{A}} = \frac{\frac{2+2+2+3}{6}}{4} = 0.37 \tag{40}$$

7 Computing $\#\mathcal{P}$ and $\#\mathcal{P}_a$ from a feature diagram

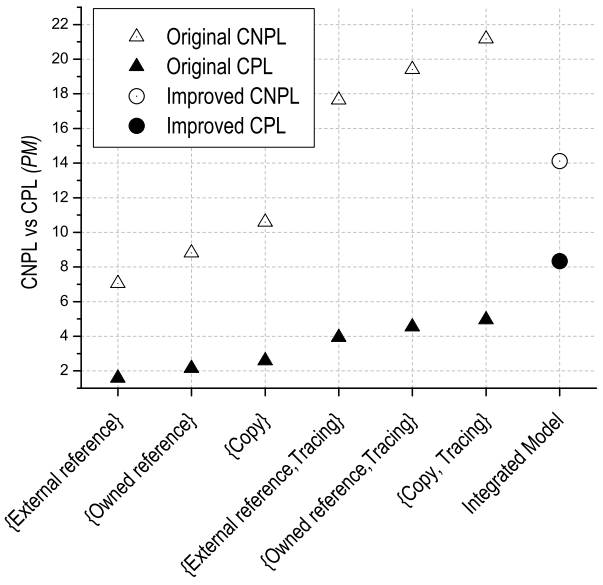
This section proposes an algorithm to compute $\#\mathcal{P}$ and $\#\mathcal{P}_a$ from a FD. A prototype implementation of the algorithm is available on:

`https://sourceforge.net/projects/commonality-spl`

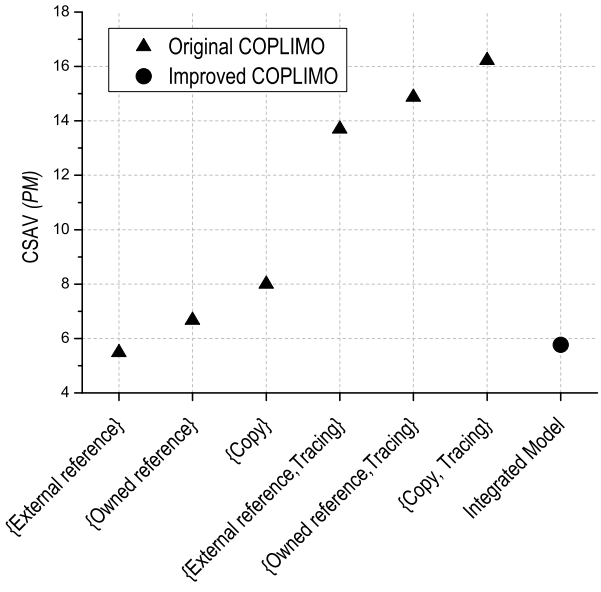
Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109



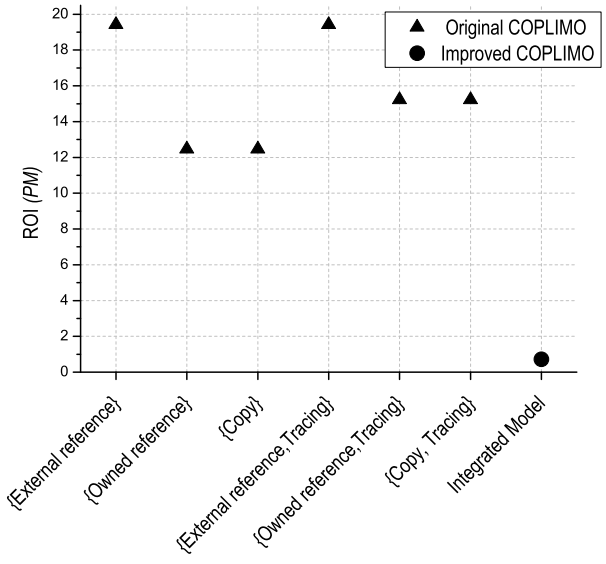
(a) CCAB



(b) CNPL and CPL



(c) CSAV



(d) ROI

Figure 4: Comparison of the estimations produced by the original COPLIMO and our improved model

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

7.1 Making uniform the child dependencies

In order to simplify the specification of the algorithm that computes $\#\mathcal{P}$ and $\#\mathcal{P}_a$, we will require all the children of every node have the same type of dependency on their parent. For instance, the FD in Figure 1 cannot be directly introduced into the algorithm because the node List has two different types of child dependencies: *mandatory* (Ownership) and *optional* (Tracing). In such cases, the FD will have to be preprocessed by using the algorithm 1, which makes uniform the child dependencies by introducing mandatory auxiliary nodes that substitute *optional*, *alternative* and *or* nodes whenever it is necessary. Note that algorithm 1 does not change the values of $\#\mathcal{P}$ and $\#\mathcal{P}_a$ of the original diagram. For example, Figure 5 depicts the transformation of the non–uniform FD in Figure 1 by introducing the auxiliary node aux_1.

Algorithm 1: MakeUniform(FD): FD

```
Data: the input FD is a feature diagram with non–uniform dependencies
Result: the output FD only includes uniform dependencies
begin
    j ← 1
    forall the non–leaf node n of FD do
        if the children of n have different types of dependencies then
            forall the group g_i of children of n do
                // g_i may be: (i) a grouped set of alternative/or children, (ii) a
                   single optional child
                d ← dependency of g_i on n
                substitute g_i for aux_j in FD
                dependency of aux_j on n is mandatory
                dependency of g_i on aux_j is d
                j ← j + 1
    return FD
```

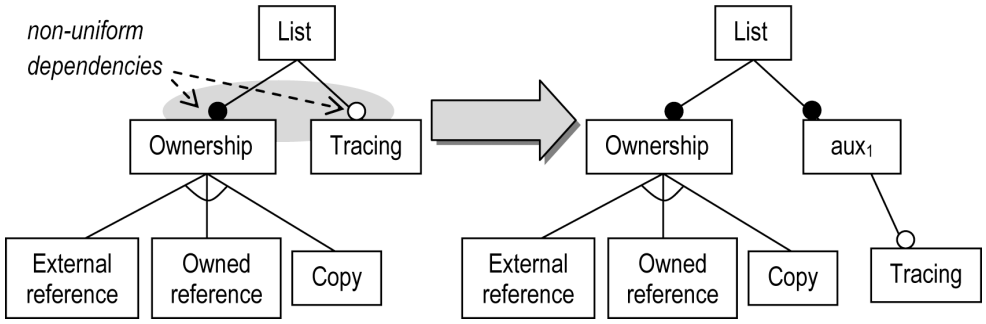


Figure 5: Making uniform the child dependencies in the list container FD

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

7.2 Computing $\#\mathcal{P}$

The *inner variability* of a node n is the number of products for a subtree with root n, and we denote it as IV_n. For instance, the inner variability of Ownership is the number of products for a FD with Ownership as root and children External reference, Owned reference and Copy. So, $\mathrm{IV}_{\text{Ownership}} = 3$.

It should be noted that $\#\mathcal{P} = \mathrm{IV}_r$, where r is the root of the diagram. For a leaf node l, $\mathrm{IV}_l = 1$. Table 5 shows the formulas to compute IV_n for a non-leaf node n with s children, $n_1, n_2, \ldots n_s$, of type *mandatory*, *optional*, *or* and *xor* respectively. The inner variability is calculated in a bottom–up fashion, from the leaves to the root, by using the algorithm 2 for any non-leaf node.

Type of relationship	Formula
mandatory	$\mathrm{IV}_n = \prod_{i=1}^{s} \mathrm{IV}_{n_i}$
optional	$\mathrm{IV}_n = \prod_{i=1}^{s} (\mathrm{IV}_{n_i} + 1)$
or	$\mathrm{IV}_n = \left(\prod_{i=1}^{s} (\mathrm{IV}_{n_i} + 1) \right) - 1$
alternative	$\mathrm{IV}_n = \sum_{i=1}^{s} \mathrm{IV}_{n_i}$

Table 5: Total number of products for *mandatory*, *optional*, *or* and *alternative* relationships

Algorithm 2: IV(n, IVChildList) : IV_n

Data: n is a node; IVChildList is a list that includes the inner variability of the children n_i of n
Result: IV_n is the inner variability of n
begin
  $\mathrm{IV}_n \leftarrow 1$
  if *n has "mandatory" children* **then**
    forall the *child n_i of n* **do**
      $\mathrm{IV}_n \leftarrow \mathrm{IV}_n \cdot \mathrm{IVChildList}[n_i]$
  else if *n has "optional" children* **then**
    forall the *child n_i of n* **do**
      $\mathrm{IV}_n \leftarrow \mathrm{IV}_n \cdot \left(\mathrm{IVChildList}[n_i] + 1\right)$
  else if *n has "or" children* **then**
    forall the *child n_i of n* **do**
      $\mathrm{IV}_n \leftarrow \mathrm{IV}_n \cdot \left(\mathrm{IVChildList}[n_i] + 1\right)$
    $\mathrm{IV}_n \leftarrow \mathrm{IV}_n - 1$
  else if *n has "alternative" children* **then**
    forall the *child n_i of n* **do**
      $\mathrm{IV}_n \leftarrow \mathrm{IV}_n + \mathrm{IVChildList}[n_i]$
  `// else leaf (do nothing)`
  return IV_n

For example, the inner variabilities in Figure 5 are:

1. Leaf nodes: $\mathrm{IV}_{\text{External reference}} = \mathrm{IV}_{\text{Owned reference}} = \mathrm{IV}_{\text{Copy}} = \mathrm{IV}_{\text{Tracing}} = 1$
2. Intermediate nodes:

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

(a) $\text{IV}_{\text{Ownership}} = \text{IV}_{\text{External reference}} + \text{IV}_{\text{Owned reference}} + \text{IV}_{\text{Copy}} = 3$

(b) $\text{IV}_{\text{aux}_1} = \text{IV}_{\text{Tracing}} + 1 = 2$

3. Root node: $\text{IV}_{\text{List}} = \text{IV}_{\text{Ownership}} \cdot \text{IV}_{\text{aux}_1} = 6$

Therefore $\#\mathcal{P} = \text{IV}_{\text{List}} = 6$.

7.3 Computing $\#\mathcal{P}_a$

If we just want to compute the number of products in the product line, IV is enough. However, if we want to know how many products a feature belongs to, it is necessary to introduce the concept of Contextual Variability (CV).

Back to the FD in Figure 1, suppose we have already computed the total number of products and the number of products each feature appears in. Now imagine we add a new root node called Container, with List and Stack as *or* children. Such change would increase $\#\mathcal{P}_{\text{List}}$ by two: for each valid configuration in the original FD, there is a new one with Container added, and another one with Container and Stack added. The CV of List is precisely 2, because each product in the List subtree is expanded into 2 different products in the bigger FD. Even more, the number of products each feature appears in has also doubled. Intuitively, the CV of List propagates down its whole subtree. One way to compute this CV is to mentally delete List and compute what the IV of its parent would be. The CV for a particular node n, is the number of different configurations for the product line ignoring the subtree rooted in n. Clearly, every product can be decomposed as the part that depends on the features in the subtree of n and the part that depends on the rest of the features. So, the number of products that n appears in for a particular solution set is the product of its IV and its CV.

CV is a global attribute that depends on the whole outside context of the node. Obviously, CV for the root feature is 1, since there is not context. In the general case, we take advantage of the fact that CV propagates downwards rather easily. For a non–root node n with parent p, $\text{CV}_n = \text{CV}_p \cdot \text{IV}_{p \wr n}$, where $p \wr n$ is a copy of p where the subtree under n has been deleted.

Algorithm 3 computes $\text{IV}_{p \wr n}$ by using the calculations of IV_n and IV_p made by algorithm 2. Finally, algorithm 4 computes $\#\mathcal{P}_a$ for all the nodes in the diagram by calling the algorithms 2 and 3.

Algorithm 3: TakeOut(p, IV_p, IV_n): $\text{IV}_{p \wr n}$

Data: node p is the parent of n; IV_p is the inner variability of p, IV_n is the inner variability of n
Result: $\text{IV}_{p \wr n}$ is the inner variability of $p \wr n$
begin
  if *n is a leaf* **then**
    $\text{IV}_{p \wr n} \leftarrow 1$
  else if *n has "mandatory" children* **then**
    $\text{IV}_{p \wr n} \leftarrow \text{IV}_p \div \text{IV}_n$
  else if *n has "optional" children* **then**
    $\text{IV}_{p \wr n} \leftarrow \text{IV}_p \div (\text{IV}_n + 1)$
  else if *n has "or" children* **then**
    $\text{IV}_{p \wr n} \leftarrow (\text{IV}_p + 1) \div (\text{IV}_n + 1)$
  else if *n has "alternative" children* **then**
    $\text{IV}_{p \wr n} \leftarrow \text{IV}_p - \text{IV}_n$
  return $\text{IV}_{p \wr n}$

For example, the contextual variabilities in Figure 5 are:

Draft version of the paper published in Expert Systems with Applications, doi:10.1016/j.eswa.2012.01.109

1. Root node:
   (a) $CV_{List} = 1$
   (b) $\#\mathcal{P}_{List} = IV_{List} \cdot CV_{List} = 6$
2. Intermediate nodes:
   (a) Ownership:
      i. $IV_{List \wr Ownership} = IV_{List} \div IV_{Ownership} = 2$
      ii. $CV_{Ownership} = CV_{List} \cdot IV_{List \wr Ownership} = 2$
      iii. $\#\mathcal{P}_{Ownership} = IV_{Ownership} \cdot CV_{Ownership} = 6$
   (b) aux_1:
      i. $IV_{List \wr aux_1} = IV_{List} \div IV_{aux_1} = 3$
      ii. $CV_{aux_1} = CV_{List} \cdot IV_{List \wr aux_1} = 3$
      iii. $\#\mathcal{P}_{aux_1} = IV_{aux_1} \cdot CV_{aux_1} = 3$
3. Leaf nodes:
   (a) External reference, Owned reference, Copy:
      i. $IV_{Ownership \wr External\ reference} = 1$
      ii. $CV_{External\ reference} = CV_{Ownership} \cdot IV_{Ownership \wr External\ reference} = 2$
      iii. $\#\mathcal{P}_{External\ reference} = IV_{External\ reference} \cdot CV_{External\ reference} = 2 = \#\mathcal{P}_{Owned\ reference} = \#\mathcal{P}_{Copy}$
   (b) Tracing:
      i. $IV_{aux_1 \wr Tracing} = 1$
      ii. $CV_{Tracing} = CV_{aux_1} \cdot IV_{aux_1 \wr Tracing} = 3$
      iii. $\#\mathcal{P}_{Tracing} = IV_{Tracing} \cdot CV_{Tracing} = 3$

8 Conclusions

In order to estimate the benefits of adopting a PL approach to develop a portfolio of software products, a number of economic models have been proposed. In this paper, we have justified the suitability of COPLIMO to support decision making on the incremental development of PLs.

COPLIMO estimates the payoff of a PL by analogy. It starts estimating the development costs of a particular product p and then, assuming that all the products in the scope of the PL are quite similar, it extrapolates the costs of p to the rest of the products. We have illustrated the kind of distortions that COPLIMO simplifying assumptions may produce. To avoid such distortions, we have reformulated COPLIMO to estimate the payoff of a PL by composition, i.e., by combining the costs of the assets that are used to build the products. To support our proposal, we have presented an algorithm that infers from a FD the information that our COPLIMO reformulation requires. Since our algorithm has quadratic time complexity, it scales for large FDs and allow the massive adoption of the COPLIMO improvement we propose.

Algorithm 4: $\#\mathcal{P}(n, \#\mathcal{P}\text{List})$

Data: n is a node; $\#\mathcal{P}$List is a reference parameter that stores the total number of products for each node (i.e., $\#\mathcal{P}\text{List}[n_i] = \#\mathcal{P}_{n_i}$); to compute $\#\mathcal{P}_a$ for the entire diagram, the algorithm should be called as $\#\mathcal{P}$(root node, $\#\mathcal{P}$List)
Result: $\#\mathcal{P}$List
begin
  // computing $\#\mathcal{P}$List in a bottom-up approach
  $\text{IVChildList}_n \leftarrow [\,]$
  forall the *child* n_i *of* n **do**
    $\#\mathcal{P}(n_i, \#\mathcal{P}\text{List})$
    $\text{IVChildList}_n \leftarrow \text{IVChildList}_n \cup \#\mathcal{P}\text{List}[m]$
  // updating $\#\mathcal{P}_n$ with IV_n
  $\#\mathcal{P}\text{List}[n] \leftarrow \text{IV}(n, \text{ListOfChildIVs}_n)$
  // updating $\#\mathcal{P}_n$ with $\text{IV}_{p \wr n}$
  forall the n_i *child of* n **do**
    $\text{IV}_{n \wr n_i} \leftarrow \text{TakeOut}(n, \text{IV}_n, \text{IV}_{n_i})$
    $\#\mathcal{P}\text{List}[n_i] \leftarrow \#\mathcal{P}\text{List}[n_i] \cdot \text{IV}_{n \wr n_i}$
    forall the d *descendant of* n_i **do**
      $\#\mathcal{P}\text{List}[d] \leftarrow \#\mathcal{P}\text{List}[d] \cdot \text{IV}_{n \wr n_i}$

References

[1] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.

[2] S. Cohen, Predicting when product line investment pays, Tech. rep., Software Engineering Institute (CMU/SEI-2003-TN-017) (2003).

[3] J. Greenfield, K. Short, S. Cook, S. Kent, Software Factories: Assembling Applications with Patterns Models Framworks and Tools, Wiley, 2004.

[4] B. Boehm, A. W. Brown, R. Madachy, Y. Yang, A software product line life cycle cost estimation model, in: International Symposium on Empirical Software Engineering, 2004, pp. 156–164.

[5] H. P. In, J. Baik, S. Kim, Y. Yang, B. Boehm, A quality-based cost estimation model for the product line life cycle, Communications of the ACM 49 (12) (2006) 85–88. doi:http://doi.acm.org/10.1145/1183236.1183273.

[6] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Y. Bontemps, Generic semantics of feature diagrams, Computer Networks 51 (2) (2007) 456–479, feature Interaction. doi:DOI: 10.1016/j.comnet.2006.08.008.

[7] A. Mili, S. F. Chmiel, R. Gottumukkala, L. Zhang, An integrated cost model for software reuse, in: ICSE '00: Proceedings of the 22nd international conference on Software engineering, ACM, New York, NY, USA, 2000, pp. 157–166. doi:http://doi.acm.org/10.1145/337180.337199.

[8] P. Clements, J. McGregor, S. Cohen, The structured intuitive model for product line economics, Tech. rep., CMU/SEI-2005-TR-003 (2005).

[9] J. S. Poulin, The economics of software product lines, International Journal of Applied Software Technology 3 (1) (1997) 20–34.

[10] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[11] D. D. Galorath, M. W. Evans, Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves, Auerbach Publications, 2006.

[12] D. R. Peterson, Economics of software product lines, in: F. van der Linden (Ed.), PFE, Vol. 3014 of Lecture Notes in Computer Science, Springer, 2003, pp. 381–402.

[13] J. P. Nobrega, E. S. de Almeida, S. R. de Lemos Meira, Income: Integrated cost model for product line engineering, in: SEAA, IEEE, 2008, pp. 27–34.

[14] S. B. A. B. Lamine, L. L. Jilani, H. H. B. Ghézala, Cost estimation for product line engineering using cots components, in: J. H. Obbink, K. Pohl (Eds.), SPLC, Vol. 3714 of Lecture Notes in Computer Science, Springer, 2005, pp. 113–123.

[15] D. Ganesan, D. Muthig, K. Yoshimura, Predicting return-on-investment for product line generations, Software Product Line Conference, International 0 (2006) 13–22. doi:http://doi.ieeecomputersociety.org/10.1109/SPLC.2006.31.

[16] Y. Chen, G. C. Gannod, J. S. Collofello, A software product line process simulator, Software Process: Improvement and Practice 11 (4) (2006) 385–409.

[17] K. Czarnecki, U. Eisenecker, Generative Programming: Methods Tools and Applications, Addison-Wesley, 2000.

[18] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature-oriented domain analysis (foda) feasibility study, Tech. rep., CMU/SEI-90-TR-21 (1990).

[19] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh, Form: A feature-oriented reuse method with domain-specific reference architectures, Ann. Software Eng. 5 (1998) 143–168.

[20] M. Griss, J. Favaro, M. Alessandro, Integrating feature modeling with the rseb, in: 5th International Conference on SoftwareReuse, IEEE Computer Society, Washington, DC, USA, 1998, pp. 76–85.

[21] M. Eriksson, J. BÃČÂűrstler, K. Borg, The pluss approach - domain modeling with features use cases and use case realizations, in: 9th International Conference on Software Product Lines, 2005, pp. 33–44.

[22] M. Riebisch, K. Böllert, D. Streitferdt, I. Philippow, Extending feature diagrams with uml multiplicities, in: 6th Conference on Integrated Design and Process Technology, 2002.

[23] J. V. Gurp, J. Bosch, M. Svahnberg, On the notion of variability in software product lines, Working IEEE/IFIP Conference on Software Architecture 0 (2001) 45–54. doi:http://doi.ieeecomputersociety.org/10.1109/WICSA.2001.948406.

[24] A. v. Deursen, P. Klint, Domain-specific language design requires feature descriptions, Journal of Computing and Information Technology 10 (1) (2002) 1–18.

[25] H. Gomaa, Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison-Wesley, 2004.

[26] K. Pohl, G. Bockle, F. Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

[27] B. Software, Gears. http://www.biglever.com/index.html.

[28] P. Systems, pure::variants. http://www.pure-systems.com/.

[29] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, G. Saval, Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis, in: 15th IEEE International Requirements Engineering Conference, 2007, pp. 243–253.

[30] D. Benavides, S. Segura, A. Ruiz-Cortés, Automated analysis of feature models 20 years later: a literature review, Information Systems 35 (6). doi:10.1016/j.is.2010.01.001.

[31] D. Benavides, A. Ruiz-Cortés, P. Trinidad, Automated reasoning on feature models, Vol. 3520, 2005, pp. 491–503.

[32] D. Benavides, On the automated analysis of software product lines using feature models. a framework for developing automated tool support, Ph.D. thesis, University of Seville, Spain (June 2007).

[33] T. Sang, F. Bacchus, P. Beame, H. Kautz, T. Pitassi, Combining Component Caching and Clause Learning for Effective Model Counting, in: 7th International Conference on Theory and Applications of Satisfiability Testing, 2004, pp. 20–28.

[34] R. Bayardo, J. Pehoushek, Counting models using connected components, in: Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000, pp. 157–162.

[35] T. Sang, P. Beame, H. A. Kautz, Heuristics for fast exact model counting, in: 8th International Conference on Theory and Applications of Satisfiability Testing, 2005, pp. 226–240.

[36] D. Batory, Feature models, grammars, and propositional formulas, in: 9th International Conference on Software Product Lines, 2005, pp. 7–20.