Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

A Literature Review on Feature Diagram Product Counting and Its Usage in Software Product Line Economic Models

Ruben Heradio*[1], David Fernandez-Amoros†[1], Jose Antonio Cerrada-Somolinos‡[1], and Ismael Abad§[1]

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

Abstract

In software product line engineering, feature diagrams are a popular means to represent the similarities and differences within a family of related systems. In addition, feature diagrams implicitly model valuable information that can be used in economic models to estimate the cost savings of a product line. In particular, this paper reviews existing proposals on computing the total number of products modeled with a feature diagram and, given a feature, the number of products that implement it. The paper also reviews the economic information that can be estimated when such numbers are known. Thus, this paper contributes by bringing together previously-disparate streams of work: the automated analysis of feature diagrams and economic models for product lines.

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, a collection of artifacts that have been designed specifically for use across the portfolio [1].

There are well documented examples of cost reduction, shorter development times, and quality improvement achieved by introducing the PL paradigm in industry [2]. Nevertheless, the PL approach is not always the best economic choice for developing a family of related products. For instance, the products may be prohibitively dissimilar from each other, or the family might contain too few products to recoup the cost of developing the core assets. Decision makers must be able to estimate the costs and benefits of building and evolving a PL as compared to traditional development approaches.

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. [3] references 40 different models and analyzes 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 [4]. To overcome this situation, several economic models oriented specifically to PLs have been proposed.

*rheradio@issi.uned.es
†david@lsi.uned.es
‡jcerrada@issi.uned.es
§iabad@issi.uned.es

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

To maximize the payoffs of a PL, its domain must 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 implemented, and implemented features may never be used, causing unnecessary complexity and both development and maintenance costs [5, 6]. To avoid these serious problems, PL domains are usually modeled by means of Feature Diagrams (FDs) [7, 8].

A FD is a compact representation of the commonalities and variabilities of the products supported by a PL, i.e., a FD does not only represent a single product but a family of them. Analyzing FDs is an error–prone and tedious task, and it is infeasible to do manually with large–scale FDs. As a result, FD analysis is an active area of research in the PL community [9].

Two essential factors that are implicitly modeled in a FD and used by economic models to estimate the cost savings of a PL are: the total number of products included in a PL (i.e., $\#\mathcal{P}$) and the number of products that implement a particular feature f (i.e., $\#\mathcal{P}_f$). Unfortunately, existing proposals to infer $\#\mathcal{P}$ and $\#\mathcal{P}_f$ from a FD have a high computational cost.

This paper reviews (i) the scoping and economic information that can be inferred whenever $\#\mathcal{P}$ and $\#\mathcal{P}_f$ are known, and (ii) the existing proposals to compute $\#\mathcal{P}$ and $\#\mathcal{P}_f$ from a FD. An effective review creates a firm foundation for advancing knowledge. It facilitates theory development, closes areas where a plethora of research exists, and uncovers areas where research is needed. In order to fulfill such goals, our review follows a rigorous and auditable methodology proposed by Kitchenham [10] and Webster et al. [11].

Compared to existing literature reviews on (i) economic models for PLs [12–14] and (ii) the automated analysis of FDs [9], the main contributions of our work are:

1. It relates both research communities. To the best of our knowledge, no paper on PL economics uses FDs as a source of information (i.e., although economic models require $\#\mathcal{P}$ and $\#\mathcal{P}_f$ as input parameters, they do not consider the modeling and analysis of FDs as a way to estimate such parameters). On the other hand, FDs were originally created as a Domain Engineering tool to model software variability [5]. Later, the use of FDs has been extended to other scenarios of software production, such as model-driven development [15], feature-oriented programming [16], and software factories [17]. As FDs have become more used, a growing necessity to support their debugging has appeared [18]. So, the literature on the automated analysis of FDs has contributed with a number of operations of analysis, tools and algorithms to support the analysis process [9]. However, papers about FD analysis rarely state the economic implications of their computations. This paper points a new area of application of FD analysis: the accurate estimation of economic model inputs.

2. It is focused on product counting for FDs. It reviews in detail how existing proposals compute $\#\mathcal{P}$ and $\#\mathcal{P}_f$ (i.e., their theoretical basis), characterizes their practical limitations and identifies the main challenges to be tackled in the future.

The remainder of the paper is structured as follows: Section 2 presents the systematic method we have used to review the literature. Using an example, Section 3 defines product counting in FDs and introduces the notation that it will be used throughout the paper. Section 4 summarizes the relevance of product counting to scope a PL adequately and estimate its cost savings. Section 5 outlines the proposals to compute $\#\mathcal{P}$ and $\#\mathcal{P}_f$ from a FD, analyzing their limitations. Section 6 discusses the results of the review and describes fundamental challenges to be faced in the future. Finally, Section 7 presents the conclusions of the paper.

2 Review method

To perform our review we have followed a systematic and structured method inspired by the guidelines of Kitchenham [10] and Webster et al. [11], which involves three main phases: planning the review, conducting

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

the review, and reporting the results.

2.1 Planning the review

The steps involved in planning a review are: identification of the need for a review, development of a review protocol and validation of the protocol.

2.1.1 Identification of the need for a review

The development of a PL usually begins by analyzing its domain [19]. The purpose of *domain analysis* is to (i) select and define the domain of focus, and (ii) collect relevant domain information and integrate it into a coherent *domain model*. A FD is a widespread tool for domain modeling [5]. Unlike traditional information models, a FD does not only represent a single product but a family of them in the same diagram. So, FDs tend to be complex and analyzing them is infeasible to do manually with large–scale feature models. Hence, automated analysis of FDs is an active area of research that is gaining importance in both practitioners and researchers in the PL community.

Another essential research topic on PL engineering is the estimation of the costs and benefits of building and evolving a PL compared to traditional development approaches. After studying a number of PL economic model proposals, we found that $\#\mathcal{P}$ and $\#\mathcal{P}_f$ were key input parameters for such models (i.e., the accuracy of the economic model estimations seemed to greatly depend on the accuracy of $\#\mathcal{P}$ and $\#\mathcal{P}_f$). So, the reason for conducting this review was identifying the accuracy needs of PL economic models and how the existing research on FD analysis could help to fulfill such needs.

2.1.2 Development of a review protocol

The second step of planning a review is the development of a review protocol, which specifies the steps and procedures the researches should follow during the review. The protocol is expected to help researchers to guarantee that the results are not influenced by researchers' expectations and desires. The systematic review protocol we have used is composed of the following elements:

1. **Identifying the research questions to be answered**. We decided that the aim of our review was to answer the following Research Questions (RQs):
   - *RQ1: Why PL economic models require to know $\#\mathcal{P}$ and $\#\mathcal{P}_f$?*
   - *RQ2: How are $\#\mathcal{P}$ and $\#\mathcal{P}_f$ computed from a FD and what are the scalability limitations of such computation?*
   - *RQ3: What are the challenges to be faced in the future?*
2. **Identifying the target audience of the review**. We decided to focus this review on the following potential readers:
   (a) Domain analysts who are interested in scoping a PL adequately and estimating its payoffs.
   (b) Researchers in PL economic models, who are interested in the information that can be retrieved automatically from a FD to improve their estimations.
   (c) Researchers in the field of automated analysis of FDs.
   (d) PL tool developers, who are interested in improving their support to decision makers.

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

3. **Strategy to search for primary studies/papers**. The search strings used in this review were constructed using the following strategy:

   - Determine and include synonyms, related terms, and alternative spelling for major terms.
   - Check the keywords in any relevant papers researchers already knew and initial searches on the relevant databases.
   - Incorporate alternative spellings and synonyms using boolean $\vee$.
   - Link main terms from population, intervention, and outcome using boolean $\wedge$.

   For instance, Equation 1 represents a search string to find works on PL economic models.

$$\begin{aligned} &\text{software} \wedge \\ &\Big(\text{product} \wedge (\text{line} \vee \text{family})\Big) \wedge \\ &\Big(\text{economic model} \vee \Big((\text{cost} \vee \text{benefits} \vee \text{payoff}) \wedge \text{estimation}\Big)\Big) \end{aligned} \tag{1}$$

   We looked for papers in the following data sources:

   - Academic digital libraries: Web of Science[1], IEEExplore[2], ACM Digital Library[3], Citeseer Library[4], and SringerLink[5].
   - The Software Engineering Institute (SEI) website[6] (SEI's serial of technical reports is the main channel of grey literature in the research area reviewed in this study).
   - Proceedings of the Software Product Line Conference (SPLC), which is the main venue for PL researchers to publish their results. We searched directly into the proceedings because some early editions of this conference were not included in any of the academic digital libraries listed.

2.1.3 Validation of the review protocol

As an indication of inclusiveness, the search results were checked for four known relevant papers on PL economics (i.e., [4, 20–22]), and three known relevant papers on FD automated analysis (i.e., [9, 18, 23]). All the relevant papers were found during the search. For further assessment of the review protocol, we contacted an independent expert on systematic reviews in software engineering and requested feedback on our review protocol. The expert provided us with useful comments, which helped us to improve the protocol.

2.2 Conducting the review

2.2.1 Identification of primary studies

The papers found in all data sources were downloaded and imported into an Mendeley[7] library. Each downloaded paper had an entry containing the publication title, abstract, author(s), source and date stored. Then, each paper was assessed for inclusion as primary study based on the following criteria:

[1] http://apps.webofknowledge.com/
[2] http://ieeexplore.ieee.org/
[3] http://dl.acm.org/
[4] http://citeseer.ist.psu.edu/
[5] http://www.springerlink.com/
[6] http://www.sei.cmu.edu/
[7] http://www.mendeley.com/

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

1. A **paper on PL economics** was included if it satisfied all the following points:
   - It was oriented specifically to PLs (i.e., papers that just dealt with the economics of software reuse were discarded).
   - It presented an approach where $\#\mathcal{P}$ or $\#\mathcal{P}_f$ were required.
2. A **paper on automated FD analysis** was included if it satisfied some of the following points:
   - It proposed an approach to compute $\#\mathcal{P}$ or $\#\mathcal{P}_f$ from a FD.
   - It included a performance study of any technique to get $\#\mathcal{P}$ or $\#\mathcal{P}_f$ from a FD.

Firstly, 68 potential primary studies were selected based on careful reading of the abstracts (and conclusions where necessary) of the papers identified through all searches after removing the duplicates. Then, a second phase of filtering was performed by reading the papers completely. After this filtering, there were 45 papers left in the process. Table 1 classifies the selected primary studies per year and type of publication.

Year	Journals	Conferences and Workshops	Technical reports	Others	#Papers
1996			[24]		1
1997	[21]				1
2000		[3,25]			2
2002		[26]			1
2003		[22,27]	[28]		3
2004		[20,29–31]		[32,33]	6
2005		[18,34–38]	[4]		7
2006	[39]	[40]		[12]	3
2007		[13,41]		[42]	3
2008	[16]	[43,44]			3
2009		[14,45–47]			4
2010	[9]	[48]		[49]	3
2011	[50]	[51–54]			5
2012	[55,56]	[57]			3
					45

Table 1: Classification of papers per year and type of publication.

2.2.2 Dealing with the heterogenous notation of economic models

Available economic models for PLs are heterogenous in terms of their main characteristics and goals [14]. The methodology we adopted to deal with the diversity of the model notations follows the guideline proposed in [58]:

- Equations of all models were translated to a common lexicon. Table 2 summarizes the components of the common lexicon used in this paper, indicating their acronym, meaning and the section on this paper where they are defined.
- Concepts that were originally formulated in an informal way (i.e., just in natural language) were formalized using our common lexicon (e.g., see Section 4.3).

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

Acronym	Description	Section
$\#S$	number of elements in set S	3
AA	Assessment and Assimilation factor	4.2
AAM	Adaptation Adjustment Modifier	4.2
AFRAC(p)	proportion of p that is composed of whitebox reused assets	4.2
CNPL($\mathcal{P}$)	Cost of building $\mathcal{P}$ under a Non-PL approach, i.e., each product individually	4.2
CPL($\mathcal{P}$)	Cost of building $\mathcal{P}$ under a PL approach	4.2
CSAV($\mathcal{P}$)	Cost SAVings of building $\mathcal{P}$ under a PL approach versus making the products individually	4.2
DOR	Degree Of Reuse	4.3
$\mathcal{F}$	set of all features; a particular feature is denoted by f	3
$\mathcal{P}$	set of all products; a particular product is denoted by p	3
$\mathcal{P}_f$	subset of $\mathcal{P}$ composed of the products that include the feature f	3
PFRAC(p),	proportion of p that is unique	4.2
$PM^{COCOMO\ II}(p)$	COCOMO II estimation of the effort measured in Person-Months that it will take to develop product p	4.2
RCR	Relative Cost of Reuse	4.1
RCWR	Relative Cost of Writing for Reuse	4.1
RFRAC(p)	proportion of p that is composed of black-box reused assets	4.2

Table 2: Common lexicon for PL economic models.

3 Products modeled with a FD

Since the first FD language was proposed by the FODA methodology in 1990 [7], a number of extensions and alternative languages have been devised to model variability in PLs:

1. As part of the following methods: FORM [59], FeatureRSEB [60], Generative Programming [5], PLUSS [61].

2. In the work of the following authors: Riebisch et al. [62], van Gurp et al. [63], van Deursen et al. [64], Gomaa [65], Pohl [19].

3. As part of the following tools: gears[8] and pure::variants[9].

Fortunately, the equivalence among most variability notations has been demonstrated in [66].

In this paper, we will use the FD notation proposed by Czarnecki et al. [67]. In addition, this section introduces the notation for product counting that will be used throughout the paper. The notation is presented using a simplified version of a PL example proposed in [5]. The aim of the example is to develop a portfolio of list containers. The PL scope is modeled with the FD in Figure 1.

A FD is a hierarchically arranged set of features, i.e., a tree $\mathbb{T}$ that represents the commonalities and variabilities of the products supported by a PL. Figure 1 includes three kinds of feature relations: mandatory (pointed by simple edges ending with a filled circle; i.e., all Lists in the portfolio have a Ownership feature), optional (pointed by simple edges ending with an empty circle; i.e., a List may have the Tracing

[8] BigLever Software: `http://www.biglever.com/`
[9] Pure Systems: `http://www.pure-systems.com/`

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

feature) and cardinality group (pointed by edges connected by an arc annotated with a <*low*–*high*> label; i.e., External reference, Owned reference and Copy are the mutually exclusive values for Ownership). The cardinality group label indicates that every product has to implement at least *low* and at most *high* of the Ownership child features. In fact, standard FODA cardinalities are special cases of the group cardinality relation. Table 3 outlines the equivalences between standard FODA cardinalities and group cardinality. 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 not 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.

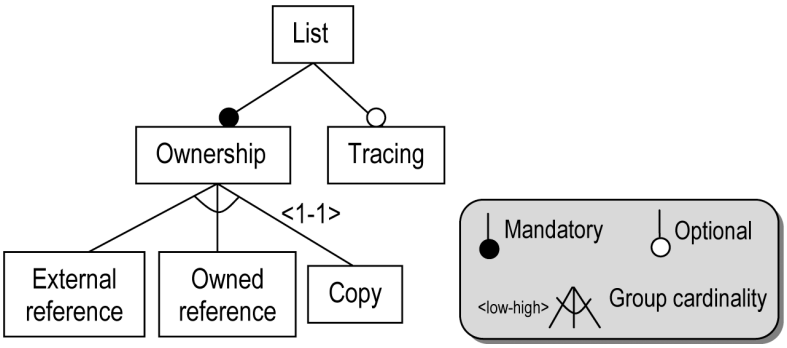

Figure 1: FD for a family of list containers

FODA cardinality	Group cardinality equivalence
Mandatory	<1-1>
Optional	<0-1>
And	<n-n>
Or	<1-n>
Xor	<1-1>

Table 3: Equivalences between standard FODA cardinalities and group cardinality.

In addition, the scope of a FD can be narrowed by adding textual constraints $\mathbb{L}$ written in propositional logic. For instance, let us suppose features Copy and Tracing are incompatible. Products implementing both features Copy and Tracing can be excluded from the scope by adding the following constraint:

$$\neg(\text{Copy} \wedge \text{Tracing})$$

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

Schobbens et al. [66] distinguish between primitive and compound features. Primitive features are "features" that are of interest *per se*, and that will influence the final product. On the contrary, compound features are just intermediate nodes used for decomposition purposes. In $\mathbb{T}$, primitive and compound features are depicted by leaves and intermediate nodes, respectively. In our example:

1. External reference, Owned reference, Copy and Tracing are primitive features.
2. List and Ownership are compound features.

For the sake of clarity, we will keep the notation for equations as concise as possible by just representing primitive features. Hence, the set $\mathcal{F}$ of features is given by equation 2.

$$\mathcal{F} = \{\text{External reference, Owned reference, Copy, Tracing}\} \tag{2}$$

A product p is denoted by the set of features that it implements (e.g., the product $p = \{\text{External reference, Tracing}\}$ includes the features External reference and Tracing).

$\mathcal{P}$ denotes the set of products included in the portfolio. In the example, $\mathcal{P}$ is given by equation 3, which excludes:

1. Products specified by invalid feature combinations; e.g., $\{\text{External reference, Copy}\} \notin \mathcal{P}$ because External reference and Copy are alternative features and consequently cannot be part of the same product.
2. Products that violate $\mathbb{L}$; e.g., $\{\text{Copy, Tracing}\} \notin \mathcal{P}$ because it violates the textual constraint $\neg(\text{Copy} \wedge \text{Tracing})$.

$$\begin{aligned} \mathcal{P} = \{ & \{\text{External reference}\}, \{\text{Owned reference}\}, \\ & \{\text{Copy}\}, \{\text{External reference, Tracing}\}, \\ & \{\text{Owned reference, Tracing}\} \} \end{aligned} \tag{3}$$

$\mathcal{P}_f$ is the subset of $\mathcal{P}$ composed of the products that implement the feature f. For instance, Equation 4 expresses $\mathcal{P}_{\text{Tracing}}$.

$$\begin{aligned} \mathcal{P}_{\text{Tracing}} = \{ & \{\text{External reference, Tracing}\}, \\ & \{\text{Owned reference, Tracing}\} \} \end{aligned} \tag{4}$$

We denote the cardinality of a set S by $\#S$ (e.g., $\#\mathcal{P}_{\text{Tracing}} = 2$ because $\mathcal{P}_{\text{Tracing}}$ has 2 products). This paper is focused on the importance and approaches to count the total number of products in a PL (i.e., $\#\mathcal{P}$) and the number of products that implement a feature f (i.e., $\#\mathcal{P}_f$).

4 Product counting usage in economic models

This section answers the first research question of our review, i.e., *Why PL economic models require to know $\#\mathcal{P}$ and $\#\mathcal{P}_f$?*. Table 4 outlines the dependencies between (i) the measures that will be presented in this section and (ii) $\#\mathcal{P}$ and $\#\mathcal{P}_f$ (e.g., the calculation of *Payoff Threshold of a Feature* requires to know $\#\mathcal{P}_f$, but not $\#\mathcal{P}$).

Results from the summarization process will be presented along with some contributions towards the formalization of models that were presented in textual form in their original research work.

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

	$\#\mathcal{P}$	$\#\mathcal{P}_f$
Payoff Threshold of a Feature [21,34,35,50]	–	√
Cost Savings of a PL [4,12,20,24,31,33,39,40,43,51,52,55–57]	√	–
Degree of Reuse throughout a PL [22,28]	–	√
Homogeneity of a PL [4]	√	√
Variability Factor of a PL [36]	√	–

Table 4: Economic measure dependencies on $\#\mathcal{P}$ and $\#\mathcal{P}_f$.

4.1 Payoff threshold of a feature

Economic models proposed in [12,20,21,34,43] use two abstractions, called the Relative Cost of Reuse (RCR) and the Relative Cost of Writing for Reuse (RCWR). RCR represents the proportion of the effort that it takes to reuse software compared to the cost normally incurred to develop it for one-time use. For instance, a feature has RCR = 0.2 if it can be reused for only 20% of the cost of implementing it. On the other hand, RCWR represents the proportion of the effort that it takes to develop reusable software compared to the cost of writing it for one-time use. For instance, if it costs an additional 50% effort to create a feature for reuse (i.e., it is necessary a more generic design, additional documentation...) then RCWR = 1.5.

Poulin [21] defines a metric called *payoff threshold*, which is also used in [34,35,50]. Payoff threshold shows how many times a feature has to be reused before the investment made to develop the feature is recovered. The payoff threshold of a feature f is calculated by equation 5.

$$\text{Payoff Threshold}_f = \frac{\text{RCWR}_f}{1 - \text{RCR}_f} \tag{5}$$

Product counting for FDs helps to detect scope flaws in the early stages of development, since a feature f causes a scope problem whenever Equation 6 holds. Note that, although all products modeled by a FD are not usually of commercial interest, $\#\mathcal{P}_f$ is an upper bound of its real reusability (i.e., real reusability$_f$ $\leq \#\mathcal{P}_f$). Equation 6 is violated whenever $\#\mathcal{P}_f$ is so small that f cannot be reused enough to recover the investment to develop it and make it easy to reuse throughout the product line. Therefore, its real reusability neither will reach the payoff threshold (i.e., as $\#\mathcal{P}_f <$ payoff threshold, then real reusability$_f <$ payoff threshold$_f$).

$$\#\mathcal{P}_f < \text{Payoff Threshold}_f \tag{6}$$

4.2 Cost savings of a PL

Economic models SIMPLE [4,31,33], COPLIMO [20,39,51,55,56] and others [52,57] estimate with Equation[10] 7 the cost savings (CSAV) for building a family of products as a PL (CPL) versus building them as stand-alone products (CNPL).

$$\text{CSAV}(\mathcal{P}) = \text{CNPL}(\mathcal{P}) - \text{CPL}(\mathcal{P}) \tag{7}$$

SIMPLE is agnostic with respect to the implementation of the cost functions CPL and CNPL. In contrast, COPLIMO approximates CSAV by assuming that all products have similar size. Hence, Equation 7 is

[10]This subsection presents SIMPLE and COPLIMO equations formulated in our common lexicon. The translation from their original formulation to the common lexicon is described in 8 (Sections 8.1 and 8.2).

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

simplified by Equation 8.

$$\mathrm{CSAV}(\mathcal{P}) \approx \mathrm{CSAV}(\#\mathcal{P}) = \mathrm{CNPL}(\#\mathcal{P}) - \mathrm{CPL}(\#\mathcal{P}) \tag{8}$$

COPLIMO computes CNPL and CPL with equations 9 and 10:

1. CNPL($\#\mathcal{P}$) is calculated by estimating the costs of developing a single product and multiplying such costs by $\#\mathcal{P}$. $\mathrm{PM}^{\text{COCOMO II}}(p)$ stands for the COCOMO II [68] estimation of the effort, measured in Person-Months (PM), necessary to develop a product p. In COCOMO II, as happen in many other methods, the estimation of software development effort relies on the previous estimation of software size [69].

$$\mathrm{CNPL}(\#\mathcal{P}) = \#\mathcal{P} \times \mathrm{PM}^{\text{COCOMO II}}(p) \tag{9}$$

2. CPL($\#\mathcal{P}$) is calculated by estimating the costs of developing a single product, making its common features reusable throughout the PL and reusing them to build the rest of the products (i.e., $\#\mathcal{P} - 1$ times). RCR is decomposed in PFRAC, RFRAC and AFRAC, which represent the proportions of a product that are unique, composed of black-box reused assets and composed of white-box reused assets[11], respectively. 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). COPLIMO assumes that all products have the same PFRAC, RFRAC and AFRAC.

$$\mathrm{CPL}(\#\mathcal{P}) = \begin{cases} \mathrm{CNPL}(1) \times \Big(\mathrm{PFRAC}(p) + \mathrm{RCWR}(p) \times \\ \quad \times \big(\mathrm{RFRAC}(p) + \mathrm{AFRAC}(p)\big)\Big) & \text{if } \#\mathcal{P} = 1 \\ \mathrm{CPL}(1) + (\#\mathcal{P} - 1) \times \mathrm{CNPL}(1) \times \big(\mathrm{PFRAC}(p) + \\ \quad + \mathrm{RFRAC}(p) \times \frac{\mathrm{AA}}{100} + \mathrm{AFRAC}(p) \times \mathrm{AAM}\big) & \text{if } \#\mathcal{P} > 1 \end{cases} \tag{10}$$

Other authors [12, 24, 40, 43] base their proposals on SIMPLE/COPLIMO and compute PL costs savings by using $\#\mathcal{P}$.

4.3 Degree of reuse throughout a PL

In order to maximize the PL payoffs, the assets that implement the common features of the products should be reused as much as possible. Regarding this issue, Cohen [28] proposes the measure Degree of Reuse (DOR) as the portion of a complete product that is made by common assets; e.g., a DOR of 0.25 means that given a typical PL member, 25% of it implements common features, and 75% of it implements unique features. Since Cohen "just expresses DOR in English", we have formalized such concept with Equation 11, where:

1. Size(f) is the size of the software that implements the feature f (a number of techniques to estimate software size are presented in [70]).

2. The dividend is the size of all the software encompassed by the PL, i.e., the size of all the products. Such size is calculated indirectly multiplying the size of the software that implements every feature (i.e., Size(f)) by the number of times that software is reused (i.e., $\#\mathcal{P}_f$).

[11] 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*.

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

3. The numerator is the size of the all software encompassed by the PL that implements common features (a feature f is common if it is implemented by more than one product, i.e., $\#\mathcal{P}_f > 1$).

$$\text{DOR}(\mathcal{P}) = \frac{\sum_{f \in \mathcal{F} | \#\mathcal{P}_f > 1} (\text{Size}(f) \times \#\mathcal{P}_f)}{\sum_{f \in \mathcal{F}} (\text{Size}(f) \times \#\mathcal{P}_f)} \tag{11}$$

In addition, to estimate the PL cost savings Peterson [22] defines the commonality parameter $\tilde{w}$, which is equivalent to DOR[12].

4.4 Homogeneity of a PL

SIMPLE defines a measure, named *homogeneity*, that characterizes how homogeneous the products are. Homogeneity is calculated by Equation 12 and varies from 0 to 1, where 0 indicates that the products are all totally unique and 1 indicates that there is only one product. According to Clements et al. [4], the higher the similarity between the products, the higher the degree of reuse and the shorter the time to market for the PL.

$$\text{Homogeneity}(\mathcal{P}) = \frac{\sum_{f \in \mathcal{F}} \frac{\#\mathcal{P}_f}{\#\mathcal{P}}}{\#\mathcal{F}} \tag{12}$$

4.5 Variability factor of a FD

The variability factor of a FD is a value between 0 and 1 that is computed by Equation 13. The smaller the ratio the more restrictive is the FD and vice-versa. The relevance of computing the variability factor has been related to decision-making strategies for adopting the PL approach [36].

$$\text{Variability}(\mathcal{P}) = \frac{\#\mathcal{P}}{2^{\#\mathcal{F}}} \tag{13}$$

5 Computing the number of products modeled with a FD

This section answers the second research question of our review, i.e., *How are $\#\mathcal{P}$ and $\#\mathcal{P}_f$ computed from a FD and what are the scalability limitations of such computation?*.

There are two main approaches to perform the automated analysis of FDs:

1. In the purely logic approach, the whole model is translated to propositional logic and then some standard procedure is applied [18,26,32,36–38,42,44,71]. In particular, $\#\mathcal{P}$ and $\#\mathcal{P}_f$ can be computed using #SAT technology [25, 29, 30]. Section 5.1 summarizes how FDs are translated to propositional logic. Afterwards, section 5.2 reviews the theoretical basis of #SAT technology.

2. In the hybrid approach, $\mathbb{L}$ is firstly processed and then some *ad-hoc* treatment is run for $\mathbb{T}$ [45,47].

Table 5 sums up the main features supported by each approach.

[12] The equivalence is described in 8 (Section 8.3).

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

	#SAT technology	Hybrid approaches	
	[25,29,30]	[47]	[45]
$\#\mathcal{P}$	$\surd$	$\surd$	$\surd$
$\#\mathcal{P}_f$	Just theoretically	–	$\surd$
Group cardinality	$\surd$	–	$\surd$
Large–scale FDs	Just for standard FODA cardinality	Just for standard FODA cardinality	–

Table 5: Features supported by approaches to analyze FDs.

5.1 Translating $\mathbb{T}$ to propositional logic

This section sketches the translation of the FD tree structure $\mathbb{T}$ into a logic formula following the directions in [72]. The resulting formula will be in Conjunctive Normal Form (CNF), that is, a conjunction of disjunctions of literals (a literal is a proposition or its negation). Such disjunctions are called clauses, so a CNF formula is a conjunction of clauses.

Suppose the FD in Figure 2. There is a root node n (i.e., A), with s children (i.e., B, C, D, E) and cardinality <*low*–*high*> (i.e., <2–3>). The relations between features are dealt with $s+1$ clauses: there is a clause to express that node n is *true*. Also, each child implies the parent node n. Figure 2 is encoded with equation 14.

$$\begin{aligned} &A \wedge (B \rightarrow A) \wedge (C \rightarrow A) \wedge (D \rightarrow A) \wedge (E \rightarrow A) \equiv \\ &A \wedge (\neg B \vee A) \wedge (\neg C \vee A) \wedge (\neg D \vee A) \wedge (\neg E \vee A) \end{aligned} \tag{14}$$

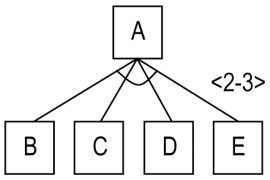

Figure 2: A very simple FD

For the cardinality restriction, we treat the *low* and *high* restrictions separately. Saying that at least *low* children have to be present in a product is equivalent to say that at most $s - low$ children can be excluded (i.e., in the logical formula no more than $s - low$ can be *false*). Which means that as soon as $s - low + 1$ literals are selected, at least one of them must be *true* (this constraint is a clause). So, the *low* restriction is equivalent to the conjunction of all possible clauses obtained by choosing $s - low + 1$ children of n. For instance, in figure 2 the *low* limit is encoded by equation 15.

$$(B \vee C \vee D) \wedge (B \vee C \vee E) \wedge (B \vee D \vee E) \wedge (C \vee D \vee E) \tag{15}$$

The *high* restriction is somewhat easier. Since in a set of $high + 1$ children at least one of them has to be *false*, we just compute all the sets of children of size $high + 1$ and add a clause with all the set members negated. See equation 16.

$$\neg(B \wedge C \wedge D \wedge E) \equiv \neg B \vee \neg C \vee \neg D \vee \neg E \tag{16}$$

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

To sum up, figure 2 is encoded by formula 17.

$$\begin{aligned}&A \wedge (\neg B \vee A) \wedge (\neg C \vee A) \wedge (\neg D \vee A) \wedge \\ &(\neg E \vee A) \wedge (B \vee C \vee D) \wedge (B \vee C \vee E) \wedge \\ &(B \vee D \vee E) \wedge (C \vee D \vee E) \wedge \\ &(\neg B \vee \neg C \vee \neg D \vee \neg E)\end{aligned} \tag{17}$$

As shown in this example, the number of clauses to encode the diagram grows quickly with s. In particular, the codification of the *low* restriction produces a combinatorial explosion, i.e., the number of clauses is $\binom{s}{s-low+1}$. Therefore, the #SAT technology described in the following section does not scale for FDs with any kind of group cardinality. Fortunately, standard FODA cardinalities (i.e., *and, or, alternative, mandatory* and *optional*) can be translated to propositional logic more efficiently [18] and, consequently, #SAT technology scales for such subset of FDs.

5.2 Model counting for propositional logic

In propositional logic, a SAT-solver is a program that tries to determine if a formula is satisfiable (i.e., if there is an assignment for the variables in the formula that make the formula *true*). This problem is known to be NP–complete [73].

A model counter is a program that tries to determine how many models a formulas has (i.e., how many satisfying assignments). Therefore, a model counter calculates $\#\mathcal{P}$ if it receives as input the formula that encodes a FD.

Model counters are usually built on top of SAT–solvers (i.e., they are #SAT–solvers). This section describes the main features of two state-of-the-art model counters: relsat [25] and cachet [29, 30].

Let us review the way these model counters work for a particular formula ϕ. We will assume (without loss of generality) that formula ϕ in CNF. Both relsat and cachet exploit two techniques named component decomposition and DPLL [74] to perform model counting.

Let us denote the number of satisfying assignments or models for formula ϕ as $\#\phi$. The rationale for component decomposition is that if a formula ϕ can be decomposed into independent connected components representing subformulas (i.e., each component comprises one or more clauses) $\phi_1, \phi_2, \ldots \phi_n$, then $\#\phi = \#\phi_1 \cdot \#\phi_2 \cdot \ldots \cdot \#\phi_n$. Therefore, the problem can be recursively reduced into simpler ones. Of course, when connected components have been identified, something else is needed to keep on simplifying. This something else is DPLL, which helps breaking a connected component ϕ_i into several ones by means of a backtracking search in which a variable x is assigned truth values and substituted for in the original formula. The resulting formulas in each branch, $\phi\,|_{x=\text{false}}$ and $\phi\,|_{x=\text{true}}$ are called residual formulas. This process goes on until the residual formula is either satisfied or a conflict is discovered. A conflict occurs when the partial assignment of variables in this search state renders the residual formula unsatisfiable (at least one of the clauses evaluates to *false*, or as it is often said, the formula contains an empty clause). Anyhow, unsatisfiable formulas are assigned zero models, and satisfied ones are assigned one model. Backtracking is then performed to continue the search. This process is outlined in Algorithm 1[13].

It is worth noting that DPLL alone is powerful enough to do some primitive model counting, but it is not very efficient. To overcome DPLL limitations, both relsat and cachet put into play different heuristics, but what makes cachet stand out is its use of component caching. Once the models of a component have been counted, this information is stored in a cache, so that if the same component appears later on during the DPLL search, the result is retrieved from the cache, instead of being recomputed. Component caching

[13] Note that the algorithm output is *model probability*. Thus, the total number of models will be: model probability $\cdot 2^{\text{number of variables of } \phi}$

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

Algorithm 1: Basic DPLL for model counting

#DPLL(ϕ): model probability
Data: ϕ is a propositional logic formula in CNF
Result: model probability = $\frac{\text{number of models of } \phi}{(2^{\text{number of variables of } \phi})}$
begin
  apply UP to ϕ;
  if *Conflict Identified* **then**
    return 0
  else if ϕ *is Satisfied* **then**
    return 1
  else
    select an unassigned variable x in ϕ;
    return $\frac{1}{2}\left(\#\text{DPLL}(\phi\ |_{x=\text{false}}) + \#\text{DPLL}(\phi\ |_{x=\text{true}})\right)$

not only saves time by avoiding repeated calculations; component caching actually prunes the DPLL search tree, since the next variable to branch on is never chosen from the cached components.

Relsat does not cache components, instead, it jumps from one component to another to try to solve unsatisfiable components first. If it finds such a conflict, there is no need to solve the other components at that level of the search, since the product would be zero anyway, and it is correct to backtrack and keep searching elsewhere.

Relsat and cachet compute efficiently $\#\mathcal{P}$. Nevertheless they do not calculate $\#\mathcal{P}_f$, which to the extent of our knowledge is a common limitation of all #SAT solvers[14].

Although #SAT solvers do not compute $\#\mathcal{P}_f$ *directly*, they can be used to calculate it *indirectly*. For instance, Kübler et al. [48] propose to compute $\#\mathcal{P}_f$ by calling a #SAT solver $\#\mathcal{F} + 1$ times, once to get $\#\mathcal{P}$ (i.e., #SAT(ϕ)), and $\#\mathcal{F}$ times to get $\#\mathcal{P}_f$ for each feature f (i.e., #SAT($\phi \wedge f$)). Unfortunately, such approach is computationally very expensive and does not scale except for the smallest FDs.

5.3 Hybrid approaches

This section reviews hybrid approaches for FD model counting in which $\mathbb{T}$ and $\mathbb{L}$ are treated separately. These systems rely on the tree structure of a FD for good performance, so they are not usually designed to gracefully handle the textual constraints, although some preprocessing may help. For instance, Czarnecki et al. [41] propose to integrate the textual constraints into the tree. The algorithm is not always able to integrate all the constraints, but a partial result may be of help to a hybrid system in some cases. Complementarily, [49] proves that all textual constraints can be eliminated at the price of building a new set of features. Unfortunately, the approach is theoretical since the basis for an implementation are laid out.

The analysis tool in the SPLOT website [47] uses Reduced Ordered Binary Decision Diagrams (ROBDDs) [75] to compute $\#\mathcal{P}$. ROBDDs are just a special case of Binary Decision Diagrams (BDDs), so we will refer to them simply as BDDs. The time complexity to compute $\#\mathcal{P}$ from a BDD with $\#u$ nodes is $O(\#u)$. Unfortunately, BDDs hinge on a fixed order in which the boolean variables are inspected. In general, the chosen variable ordering makes a significant difference to the size of the BDD representing a given boolean formula [76]. Although finding an ordering that produces an optimal BDD (i.e., a BDD with the minimal $\#u$) is an NP-complete problem [77], there are heuristics which usually produce a fairly good ordering. For instance, the SPLOT tool represents $\mathbb{L}$ as a BDD and uses a heuristic that takes into account the whole model (i.e., $\mathbb{T}$ and $\mathbb{L}$) to determine the order of the variables.

[14] In [29], the authors of cachet sketch a pseudocode to compute the marginal probabilities of the variables of a boolean formula (i.e., the probability that a certain variable f is present in a random product), which, in our notation would correspond to $\frac{\#\mathcal{P}_f}{\#\mathcal{P}}$. Unfortunately, the implementation provided by the authors does not allow this computation.

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

Mendonça [78], the author of SPLOT, makes a big selling point of the space gains of the BDD vs. DPLL, however, DPLL being a backtracking search, it only keeps in memory the current stack of search nodes. Even if the BDD graph is more compact than the corresponding DPLL search tree (although the latter is never explicitly constructed) the BDD graph has to be traversed, which is not unlike performing a DPLL search. The main difference is that only textual constraints are represented in the BDD, so the boolean formula is much simpler than in purely logic systems, which have to bear with the diagram translation as well. Regarding the input, although standard FODA is extended in Mendonça's metamodel [46] to support group cardinality, the SPLOT tool only supports standard *mandatory/optional*, *or* and *xor* cardinalities.

Pohl et al. [53] have carried out a performance comparison of algorithmic approaches for automated analysis operations on FDs. In particular, the capability of SAT and BDD solvers to compute $\#\mathcal{P}$ is compared, and it seems that BDD technology performs better than SAT for large models. Nevertheless, since Pohl does not include #SAT solvers in the comparison, and SAT solvers are tuned to find just a satisfying variable assignment, not to compute the number of all satisfying assignments, Pohl's study is not conclusive.

In a similar way that happens with #SAT technology: (i) there are BDD tools that efficiently compute $\#\mathcal{P}$, but (ii) to the extent of our knowledge, there is no approach to calculate $\#\mathcal{P}_f$ using BDD technology. In contrast, Fernandez-Amoros et al. [45] propose an algorithm specifically oriented to FDs that calculates $\#\mathcal{P}$ and $\#\mathcal{P}_f$. It is noteworthy that although the algorithm has polynomial time complexity for $\mathbb{T}$, it is always exponential in the number of clauses of the textual constraints. So, although the algorithm scales for FDs with any size of $\mathbb{T}$, when $\mathbb{L}$ is big the algorithm is useless.

6 Future challenges

This section answers the third research question of our review, i.e., *What are the challenges to be faced in the future?*

6.1 Selecting the approach that fits better for a given FD

As far as the strengths and weaknesses of the product counting approaches are concerned, component-based model counters such as cachet and relsat are expected to be very efficient when the textual constraints present independent clauses (clauses with no variables in common) since DPLL can efficiently cut the translated formula into many different connected components, and very inefficient when the diagram heavily exploits group cardinality other than standard FODA, since the translation to boolean logic suffers from a combinatorial explosion. The concept of branch-width [27] accurately explains this behavior.

Comparatively, the hybrid approaches are expected to perform poorly when textual constraints are mostly independent (formulas with a low branch-width). For SPLOT, the problem is that traversing the BDD is similar to traversing a binary search tree, except for the fact that the BDD is traversed backwards starting from the *true* vertex, so conflicts are never traversed (they finish in the *false* vertex which is not traversed). This way, the number of nodes to traverse will be exponential. In addition, SPLOT does not support the the general group cardinality construction.

The worst case for cachet, as far as textual constraints are concerned, would be a diagram with only one textual clause involving all the features. In this case, if N is the number of features, time complexity would be at least $O(2^N)$, while for the hybrid approach we proposed in [45], it would be just $O(N^2)$.

The bottom line is that there are no silver bullets for this problem. Algorithms perform better for some instances and worse for others. Therefore, a precise criteria should be defined to automatically select the approach that fits better for a given FD.

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

6.2 Towards a scalable approach to compute $\#\mathcal{P}_f$

As shown in Table 5, the only implemented algorithm that supports the computation of $\#\mathcal{P}_f$ is [45]. Unfortunately, the algorithm is always exponential in the number of clauses of the textual constraints, not just in the worst case. A more scalable algorithm should be implemented for real FDs.

6.3 Improving economic estimates by means of $\#\mathcal{P}_f$

Economic models often make simplifying assumptions when $\#\mathcal{P}_f$ is unknown. Such assumptions may produce high distortions in the estimates.

For instance, in section 4.4 we presented equation 12 to calculate the homogeneity of a PL. However, for the cases where $\#\mathcal{P}_f$ is unavailable but the number of unique features $\#\mathcal{UF}$ is known (i.e., features implemented by only one product), SIMPLE [4] proposes the alternative equation 18.

$$\text{Homogeneity}(\mathcal{P}) = 1 - \frac{\#\mathcal{UF}}{\#\mathcal{F}} \tag{18}$$

Unfortunately, this measure may produce erroneous results in some scenarios. For example, consider a PL with 100 features, where every feature is included in 2 products and every product implements 10 features; although the PL is quite heterogeneous, equation 18 says that the PL is completely homogeneous (i.e., homogeneity $= 1 - \frac{0}{100} = 1$).

In the future, economic models could be reformulated for the cases where $\#\mathcal{P}_f$ is available. In particular, COPLIMO estimates PL cost savings ignoring $\#\mathcal{P}_f$ and assuming that all the products have similar size (see section 4.2). Clearly, such assumption only works for domains extremely homogeneous. In [55] a more realistic estimate is computed taking into account $\#\mathcal{P}_f$.

6.4 Interacting with other software engineering communities

Other software engineering communities also use model counting to tackle variability issues. For instance, automated configurators have been used to support product mass customization for more than thirty years [79]. They let customers select and validate a configuration on the basis of available product assets and predefined constraints for asset composition. The configuration of all but trivial systems involves considerable effort because it requires making a large number of decisions regarding which assets are going to be included/excluded in/from the system. Some approaches (e.g., [54, 80, 81] use $\#\mathcal{P}$ and $\#\mathcal{P}_f$ to alleviate such effort speeding up the process to reach a valid configuration by reducing the number of decisions to be made. That is, taking advantage of the fact that, due to the asset composition constraints, some decisions may be automatically derived from other decisions previously made. So the order in which decisions are made has a strong influence on the number of decisions required to complete a configuration.

The jointly usage of economic models and automated configurators could help users to reach not only valid products, but economically efficient ones.

7 Conclusions

In this paper, we have revised the state of the art on product counting for FDs and its applicability to economic models for PLs by running a structured literature review that encompasses 45 primary studies and outlines the main advances made to date.

It has been shown that $\#\mathcal{P}$ and $\#\mathcal{P}_f$ are essential factors to accurately estimate the payoffs of a PL and to scope its domain. Unfortunately, product counting has an high computational cost. There is no

optimal approach for the general problem (i.e., taking into account textual constraints, group cardinality...). Reviewed approaches perform better for some diagrams and worse for others. Therefore, a fundamental challenge to be tackled in the future is the automated selection of the approach that fits better for each FD.

Acknowledgment

The authors are grateful to the anonymous reviewers for their insightful feedback. We also thank Roberto Lopez Herrejon and Alexander Egyed at the Institute for Software Engineering and Automation (Johannes Kepler University of Linz, Austria) for their advice. This work has been supported by the Spanish Open University under grant 2010V/PUNED/004 and by the Comunidad de Madrid under the RoboCity2030-II excellence research network S2009/DPI-1559

References

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

[2] C. Dinnus and K. Pohl, *Software Product Line Engineering*, ch. Experiences with Software Product Line Engineering, pp. 413–434. Springer Berlin Heidelberg, 2005.

[3] A. Mili, S. F. Chmiel, R. Gottumukkala, and L. Zhang, "An integrated cost model for software reuse," in *22nd International Conference on Software Engineering*, (New York, USA), pp. 157–166, 2000.

[4] P. Clements, J. McGregor, and S. Cohen, "The structured intuitive model for product line economics," tech. rep., CMU/SEI-2005-TR-003, Software Engineering Institute, 2005.

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

[6] J. Lee, S. Kang, and D. Lee, "A comparison of software product line scoping approaches," *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, no. 5, pp. 637–663, 2010.

[7] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," tech. rep., CMU/SEI-90-TR-21, Software Engineering Institute, 1990.

[8] M. Boskovic, E. Bagheri, D. Gasevic, B. Mohabbati, N. Kaviani, and M. Hatala, "Automated staged configuration with semantic web technologies," *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, no. 4, pp. 459–484, 2010.

[9] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, vol. 35, no. 6, 2010.

[10] B. Kitchenham, "Procedures for performing systematic reviews," Tech. Rep. TR/SE-0401, Software Engineering Group. Department of Computer Science. Keele University, 2004.

[11] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS Quarterly*, vol. 26, no. 2, pp. 13–23, 2002.

[12] J. H. Wesselius, *Software Product Lines: Research Issues in Engineering and Management*, ch. Strategic Scenario-Based Valuation of Product Line Roadmaps, pp. 53–89. Springer Berlin Heidelberg, 2006.

[13] S. B. A. B. Lamine, L. L. Jilani, A. Louhichi, and H. H. B. Ghézala, "Software product line economics: a survey," in *5th Intenational Conference on Software Engineering Research and Practice*, (Las Vegas, Nevada, USA), 2007.

[14] M. S. Ali, M. A. Babar, and K. Schmid, "A comparative survey of economic models for software product lines," in *35th Euromicro Conference on Software Engineering and Advanced Applications*, (Patras, Greece), pp. 275–278, IEEE Computer Society, 2009.

[15] P. Heymans, P. Y. Schobbens, J. C. Trigaux, Y. Bontemps, R. Matulevicius, and A. Classen, "Evaluating formal properties of feature diagram languages," *IET Software*, vol. 2, no. 3, pp. 281–302, 2008.

[16] D. Batory and E. Börger, "Modularizing theorems for software product lines: The jbook case study," *Journal of Universal Computer Science*, vol. 14, no. 12, pp. 2059–2082, 2008.

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

[18] D. Batory, "Feature models, grammars, and propositional formulas," in *9th International Software Product Line Conference*, pp. 7–20, 2005.

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

[20] B. Boehm, A. W. Brown, R. Madachy, and Y. Yang, "A software product line life cycle cost estimation model," in *International Symposium on Empirical Software Engineering*, (Redondo Beach, CA, USA), 2004.

[21] J. S. Poulin, "The economics of software product lines," *International Journal of Applied Software Technology*, vol. 3, pp. 20–34, March 1997.

[22] D. R. Peterson, "Economics of software product lines," in *5th International Workshop in Product Family Engineering*, (Siena, Italy), 2003.

[23] A. van Deursen and P. Klint, "Domain–Specific Language Design Requires Feature Descriptions," *Journal of Computing and Information Technology*, vol. 10, no. 1, pp. 1–17, 2002.

[24] J. Withey, "Investment analysis of software assets for product lines," tech. rep., CMU/SEI-96-TR-010, Software Engineering Institute, 1996.

[25] R. Bayardo and J. Pehoushek, "Counting models using connected components," in *17th National Conference on Artificial Intelligence*, (Austin, Texas, USA), pp. 157–162, 2000.

[26] M. Mannion, "Using first-order logic for product line model validation," in *2nd International Software Product Line Conference*, (London, UK), pp. 176–187, Springer-Verlag, 2002.

[27] F. Bacchus, S. Dalmao, and T. Pitassi, "Algorithms and Complexity Results for #SAT and Bayesian Inference," in *44th Annual IEEE Symposium on Foundations of Computer Science*, (Cambridge, MA, USA), pp. 340–351, 2003.

[28] S. Cohen, "Predicting when product line investment pays," tech. rep., CMU/SEI-2003-TN-017, Software Engineering Institute, 2003.

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

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

[31] G. Bockle, P. Clements, J. McGregor, D. Muthig, and K. Schmid, "Calculating roi for software product lines," *IEEE Software*, vol. 21, no. 3, pp. 23–31, 2004.

[32] M. Mannion and J. Camara, "Theorem proving for product line model verification," in *Software Product–Family Engineering* (F. van der Linden, ed.), vol. 3014 of *Lecture Notes in Computer Science*, pp. 211–224, Springer Berlin Heidelberg, 2004.

[33] G. Bockle, P. Clements, J. McGregor, D. Muthig, and K. Schmid, "A cost model for software product lines," in *Software Product-Family Engineering* (F. van der Linden, ed.), vol. 3014 of *Lecture Notes in Computer Science*, pp. 310–316, Springer Berlin / Heidelberg, 2004.

[34] S. B. A. B. Lamine, L. L. Jilani, and H. H. B. Ghezala, "Cost estimation for product line engineering using cots components," in *9th International Software Product Line Conference*, (Rennes, France), 2005.

[35] S. B. A. B. Lamine, L. L. Jilani, and H. H. B. Ghezala, "A software cost estimation model for a product line engineering approach: Supporting tool and uml modeling," in *3rd International Conference on Software Engineering Research, Management and Applications*, (Los Alamitos, CA, USA), 2005.

[36] D. Benavides, A. Ruiz-Cortés, and P. Trinidad, "Automated reasoning on feature models," in *17th International Conference on Advanced Information Systems Engineering*, (Porto, Portugal), pp. 491–503, 2005.

[37] K. Czarnecki and P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *International Workshop on Software Factories At OOPSLA*, (San Diego, CA, USA), 2005.

[38] D. Benavides, S. Segura, P. T. Martín-Arroyo, and A. R. Cortés, "Using java csp solvers in the automated analyses of feature models," in *Generative and Transformational Techniques in Software Engineering*, (Braga, Portugal), pp. 399–408, 2005.

[39] H. P. In, J. Baik, S. Kim, Y. Yang, and B. Boehm, "A quality-based cost estimation model for the product line life cycle," *Communications of the ACM*, vol. 49, no. 12, pp. 85–88, 2006.

[40] J. P. Nobrega, E. S. D. Almeida, and S. R. D. L. Meira, "A cost framework specification for software product lines scenarios," in *6th Workshop on Component–Based Development*, (Recife, Brazil), 2006.

[41] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," in *SPLC '07: Proceedings of the 11th International Software Product Line Conference*, (Washington, USA), p. 23–34, IEEE Computer Society, 2007.

[42] D. Benavides, *On the automated analysis of software product lines using feature models. A framework for developing automated tool support*. PhD thesis, University of Seville, Spain, June 2007.

[43] J. P. Nobrega, E. S. de Almeida, and S. R. de Lemos Meira, "Income: Integrated cost model for product line engineering," in *34th Euromicro Conference Software Engineering and Advanced Applications*, (Parma,Italy), 2008.

[44] S. Segura, "Automated analysis of feature models using atomic sets," in *1st Workshop on Analyses of Software Product Lines*, (Limerick, Ireland), pp. 201–207, 2008.

[45] D. Fernandez-Amoros, R. Heradio, and J. Cerrada, "Inferring information from feature diagrams to product line economic models," in *13th International Software Product Line Conference*, pp. 41–50, 2009.

[46] M. Mendonça, *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, 2009.

[47] M. Mendonça, M. Branco, and D. Cowan, "S.P.L.O.T. - Software Product Lines Online Tools," in *24th Conference on object oriented programming systems languages and applications*, (Orlando, Florida, USA), 2009.

[48] A. Kübler, C. Zengler, and W. Küchlin, "Model counting in product configuration," in *1st International Workshop on Logics for Component Configuration*, (Edinburgh, Scotland), July 2010.

[49] Y. Gil and S. Kremer-Davidson, "Sans Constraints? Feature Diagrams vs. Feature Models," *Lecture Notes in Computer Science*, vol. I, no. 6287, pp. 271–285, 2010.

[50] R. Heradio-Gil, D. Fernandez-Amoros, J. A. Cerrada, and C. Cerrada, "Supporting commonality-based analysis of software product lines," *IET Software*, vol. 5, no. 6, pp. 496–509, 2011.

[51] V. Nguyen, L. Huang, and B. Boehm, "An analysis of trends in productivity and cost drivers over years," in *7th International Conference on Predictive Models in Software Engineering*, (New York, NY, USA), pp. 3:1–3:10, ACM, 2011.

[52] J. Müller, "Value-based portfolio optimization for software product lines," in *15th International Software Product Line Conference*, (Munich, Germany), pp. 15–24, 2011.

[53] R. Pohl, K. Lauenroth, and K. Pohl, "A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models," in *26th IEEE/ACM International Conference on Automated Software Engineering*, (Lawrence, Kansas, USA), pp. 313–322, November 2011.

[54] S. Chen and M. Erwig, "Optimizing the product derivation process," in *15th International Software Product Line Conference*, (Munich, Germany), pp. 25–34, August 2011.

[55] R. Heradio, D. Fernandez-Amoros, L. Torre-Cubillo, and A. P. Garcia-Plaza, "Improving the accuracy of coplimo to estimate the payoff of a software product line," *Expert Systems with Applications*, vol. 39, no. 9, pp. 7919–7928, 2012.

[56] M. Gongora-Blandon, P. Picota-Cano, and C. Clunie, "Cost estimate methods comparison in software product lines (spl)," *Revista Universitaria en Telecomunicaciones, Informatica y Control*, vol. 1, no. 2, pp. 18–24, 2012.

[57] J. Gillain, S. Faulkner, P. Heymans, I. Jureta, and M. Snoeck, "Product portfolio scope optimization based on features and goals," in *16th International Software Product Line Conference*, (New York, NY, USA), pp. 161–170, ACM, 2012.

[58] W. C. Lim, "Reuse economics: A comparison of seventeen models and directions for future research," in *4th International Conference on Software Reuse*, (Los Alamitos, CA, USA), 1996.

[59] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "Form: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, pp. 143–168, 1998.

[60] M. Griss, J. Favaro, and M. Alessandro, "Integrating feature modeling with the rseb," in *5th International Conference on Software Reuse*, (Washington, DC, USA), pp. 76–85, 1998.

[61] M. Eriksson, J. Böstler, and K. Borg, "The pluss approach – domain modeling with features, use cases and use case realizations," in *Software Product Lines* (H. Obbink and K. Pohl, eds.), vol. 3714 of *Lecture Notes in Computer Science*, pp. 33–44, Springer Berlin Heidelberg, 2005.

[62] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow, "Extending Feature Diagrams with UML Multiplicities," in *6th World Conference on Integrated Design & Process Technology*, (Pasadena, California), 2002.

[63] J. van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *Working IEEE/IFIP Conference on Software Architecture*, (Amsterdam, The Netherlands), pp. 45–54, 2001.

[64] A. v. Deursen and P. Klint, "Domain-specific language design requires feature descriptions," *Journal of Computing and Information Technology*, vol. 10, no. 1, pp. 1–18, 2002.

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

[66] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps, "Generic semantics of feature diagrams," *Computer Networks*, vol. 51, no. 2, pp. 456–479, 2007.

[67] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Staged configuration using feature models.," in *3rd International Software Product Line Conference*, (Boston, MA, USA), pp. 266–283, 2004.

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

[69] S. Tunalilar and O. Demirörs, "An exploration of functional size based effort estimation models," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 3, pp. 413–429, 2011.

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

[71] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens, and 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*, pp. 243–253, 2007.

[72] A. Biere, M. J. Heule, H. van Maaren, Toby, and Walsh, *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[73] S. A. Cook, "The complexity of theorem-proving procedures," in *3rd ACM symposium on Theory of computing*, (New York, NY, USA), pp. 151–158, 1971.

[74] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[75] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[76] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[77] B. Bollig and I. Wegener, "Improving the variable ordering of obdds is np-complete," *IEEE Transactions on Computers*, vol. 45, pp. 993–1002, September 1996.

[78] M. Mendonça, *Efficient Reasoning Techniques for Large Scale Feature Models*. PhD thesis, University of Waterloo, 2009.

[79] D. Sabin and R. Weigel, "Product configuration frameworks-a survey," *IEEE Intelligent Systems*, vol. 13, pp. 42–49, July 1998.

[80] A. Nöhrer and A. Egyed, "Optimizing user guidance during decision-making," in *15th International Software Product Line Conference*, (Munich, Germany), pp. 25–34, August 2011.

[81] A. Nöhrer and A. Egyed, "C2o configurator: a tool for guided decision-making," *Automated Software Engineering*, vol. 20, pp. 265–296, June 2013.

8 Translation of economic model equations to the common lexicon

This appendix describes how the equations presented in Section 4 have been translated from their original notation to the common lexicon used in this paper.

8.1 SIMPLE

Table 6 summarizes the notational equivalences between SIMPLE parameters and our proposed common lexicon.

SIMPLE	Common lexicon
Cost savings of building a PL vs. stand-alone products	CSAV($\mathcal{P}$)
Cost of building n standalone products	CNPL($\mathcal{P}$)
Cost of building a PL	CPL($\mathcal{P}$)
NP	$\#\mathcal{P}$
Number of products satisfying R_i	$\#\mathcal{P}_f$
$\|R_T\|$	$\#\mathcal{F}$

Table 6: Equivalences between SIMPLE parameters and our proposed common lexicon.

1. The original formulation of Equation 8 is Equation 19.

$$\begin{aligned}&\text{Cost savings of building a PL vs. stand-alone products} = \\ &= \text{Cost of building } n \text{ standalone products} - \\ &- \text{Cost of building a PL}\end{aligned} \tag{19}$$

2. The original formulation of Equation 12 is Equation 20, where:

   - NP is the number of products in the product line
   - R_i is a functional requirement of a product
   - $\|R_T\|$ is the total number of different requirements

$$\text{Homogeneity} = \frac{\sum_{i=1}^{T} \frac{\text{Number of products satisfying } R_i}{\text{NP}}}{\|R_T\|} \tag{20}$$

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

8.2 COPLIMO

Table 7 summarizes the notational equivalences between COPLIMO parameters and our proposed common lexicon.

COPLIMO	Common lexicon
PLS(N)	CSAV($\#\mathcal{P}$)
PMNR(N)	CNPL($\#\mathcal{P}$)
PMNR(N)	CPL($\#\mathcal{P}$)
N	$\#\mathcal{P}$
$A \times \text{Size}^B \times \prod_i \text{EM}_i$	$\text{PM}^{\text{COCOMO II}}(p)$

Table 7: Equivalences between COPLIMO parameters and our proposed common lexicon.

1. The original formulation of Equation 9 is Equation 21, where PMR(N) and PMNR(N) stand for the PM effort to develop N products with reuse and without it (i.e., PL vs individual development).

$$\text{PLS}(N) = \text{PMNR}(N) - \text{PMR}(N) \tag{21}$$

2. The original formulation of Equation 9 is Equation 22, where:

   - 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 that account 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).

$$\text{PMNR}(N) = N \times A \times \text{Size}^B \times \prod_i \text{EM}_i \tag{22}$$

3. The original formulation of Equation 10 is Equation 23.

$$\text{PMR}(N) = \begin{cases} \text{PMNR}(1)\times & \\ \quad \times \Big(\text{PFRAC+RCWR} \times (\text{RFRAC+AFRAC})\Big) & \text{if } N{=}1 \\ \text{PMR}(1){+}(N\text{-}1) \times \text{PMNR}(1)\times & \\ \quad \times (\text{PFRAC} + \text{RFRAC} \times \frac{\text{AA}}{100} + \text{AFRAC} \times \text{AAM}) & \text{if } N > 1 \end{cases} \tag{23}$$

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

8.3 Peterson's model

Peterson [22] characterizes the PL savings estimating the redundant efforts needed to develop independently the products compared to the efforts consumed by the PL approach. According to Peterson's notation, $\mathfrak{R}$ represents the set of requirements for the PL. In the example presented in section 3, $\mathfrak{R}$ is given by Equation 24.

$$\mathfrak{R} = \{\text{External reference, Owned reference, Copy, Tracing}\} \tag{24}$$

Each one of the N products of the PL is associated with an ordinal number k. $\mathfrak{R}_k$ represents the set of requirements associated with product k. In the example, $N = 5$ and the ordinal numbers k will follow the order in which products appear in Equation 3 (e.g., $\mathfrak{R}_1 = \{\text{External reference}\}$, $\mathfrak{R}_2 = \{\text{Owned reference}\}$, ..., $\mathfrak{R}_5 = \{\text{Owned reference, Tracing}\}$).

$\mathfrak{R}$ is partitioned into disjoint subsets in such a way that all requirements in a given subset apply to the same set of products. Let $\{\pi\}$ denote the set of N-dimensional vectors that satisfy: $|\pi| > 0; \pi_k = 0$ or 1. Then a partition[15], $\mathfrak{R}_\pi \subseteq \mathfrak{R}$, is defined by Equation 25, where the complement is relative to $\mathfrak{R}$.

$$\mathfrak{R}_\pi \equiv \Big(\bigcap_{\{k|\pi_k=1\}} \mathfrak{R}_k \Big) \bigcap \Big(\bigcup_{\{k|\pi_k=0\}} \mathfrak{R}_k \Big)^C \tag{25}$$

A '1' appearing in the k^{th} position of a vector π means the k^{th} product is a member of the partition, whereas a value of '0' indicates the product is not a member of the partition. For instance, if $\pi = (0,0,0,1,1)$, $\mathfrak{R}_\pi$ represents the set of requirements shared by products 4 and 5, but not shared by products 1, 2 and 3 (i.e., $\mathfrak{R}_\pi = \{\text{Tracing}\}$). If $\pi = (1,0,0,0)$, $\mathfrak{R}_\pi$ represents the requirements unique to product 1 (i.e., $\mathfrak{R}_\pi = \emptyset$ since the feature External reference is common to products 1 and 4). Thus, although $|\{\pi\}| = 2^N - 1$, most partitions are empty. Table 8 shows the non-empty partitions in the example.

Peterson defines the following parameters for partitions:

- *weight n*: it is the number of products that are members of the partition (i.e., $n(\pi)$ is the number of '1's in the π vector). For instance, $n(0,0,0,1,1) = 2$.
- *size*: it is calculated by the demand function D_π, which estimates the size of the assets that will implement the requirements of $\mathfrak{R}_\pi$. The demand placed on a given product is given by Equation 26, where $\hat{k}$ is the unit vector whose components are all '0' except for the k^{th} component which is equal to '1' (e.g., $\hat{2} = (0,1,0,0,0)$).

$$D_k = \sum_{\{\pi\}} \hat{k} \times \pi \times D_\pi \tag{26}$$

In the independent scenario, each product group must develop, enhance, and maintain its own code base to meet the set of requirements, even though other products may share many of those requirements. Thus, the total effective demand across all products in the independent scenario is calculated by Equation 27. In the example, $\tilde{D} = 2 \cdot \text{Size(External reference)} + 2 \cdot \text{Size(Owned reference)} + 1 \cdot \text{Size(Copy)} + 2 \cdot \text{Size(Tracing)}$.

$$\tilde{D} = \sum_{k=1}^{N} D_k = \sum_{k=1}^{N} \Big(\sum_{\{\pi\}} (\hat{k} \times \pi \times D_\pi) \Big) = \sum_{\{\pi\}} \big(n(\pi) \times D_\pi \big) \tag{27}$$

[15] We follow Peterson's notation here, but $\mathfrak{R}_\pi$ (if not empty) is not really a partition of $\mathfrak{R}$ but rather a subset of the partition. The set of all the nonempty $\mathfrak{R}_\pi$ is really a partition, in the equivalence relation sense.

Draft version of the paper published in IJSEKE, Oct.2013, doi:10.1142/S0218194013500368

To estimate the PL cost savings, Peterson defines the *commonality* parameter $\tilde{w}$, which is calculated by Equation 28 and represents the fraction of demand in the independent case that is shared by multiple products. In the equation, $\{\pi\}^c$ represents the set of partitions which apply to two or more products.

$$\tilde{w} = \frac{\sum_{\{\pi\}^c} (n(\pi) \times D_\pi)}{\tilde{D}} \tag{28}$$

The formulation of Peterson's model becomes clearer in our lexicon, since the π-partition formalism is avoided. Thus, Equation 27 is translated to its equivalent Equation 29. Note that Peterson's only takes into account software size to estimate cost of making software (i.e., $D_f = \text{Size}(f)$).

$$\tilde{D} = \sum_{f \in \mathcal{F}} (\text{Size}(f) \times \#\mathcal{P}_f) \tag{29}$$

Finally, Equation 28 is translated to Equation 30, i.e., $\tilde{w}$ is what was defined as DOR in Equation 11.

$$\tilde{w} = \frac{\sum_{\{\pi\}^c} (n(\pi) \times D_\pi)}{\sum_{f \in \mathcal{F}} (\text{Size}(f) \times \#\mathcal{P}_f)} = \frac{\sum_{f \in \mathcal{F} | \#\mathcal{P}_f > 1} (\text{Size}(f) \times \#\mathcal{P}_f)}{\sum_{f \in \mathcal{F}} (\text{Size}(f) \times \#\mathcal{P}_f)} = \text{DOR}(\mathcal{P}) \tag{30}$$