Software Product Lines

Exercise 8: Development Process

ISF (TU Braunschweig) January 2026

1. **Domain and Application Engineering**

   Complete the following figure with the process steps employed in *domain* and *application engineering*.

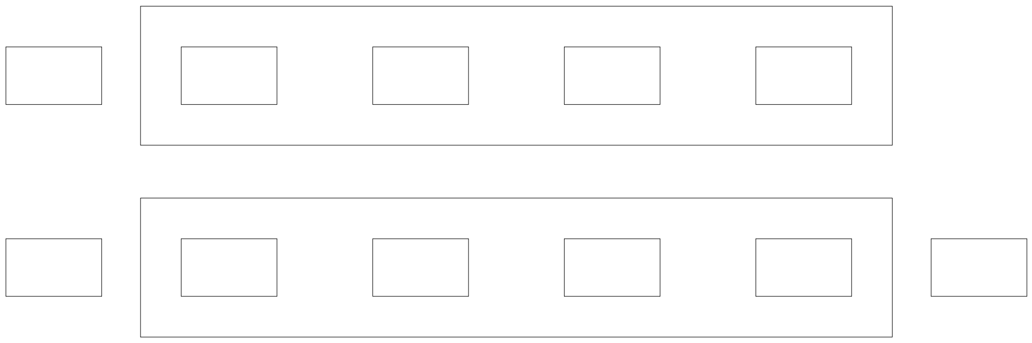
   (a) Fill in each box with the name of the corresponding step (outer and inner boxes!).

   (b) Add all missing edges / arrows.

   (c) How and where can custom development be supported?

   (d) Highlight which steps belong to the problem space and which steps belong to the solution space.

   Explain the resulting figure. In your opinion, in which activity should you invest the most efforts and why?



   (e) Besides, give examples for scenarios in which ...

   i. almost all of the effort is invested in domain engineering

   ii. almost all of the effort is invested in application engineering

   iii. the effort is balanced between domain and application engineering

2. **Adoption Strategies for Software Product Lines**

   (a) Explain three typical adoption strategies for introducing software product lines.

   (b) Recommend the most suitable strategy for each scenario given below and justify your answer.

   i. A company has developed three separate CRM systems for different regions. They share 60–70% of features but were developed independently. The company wants to unify the codebase to reduce maintenance costs while keeping region-specific capabilities.

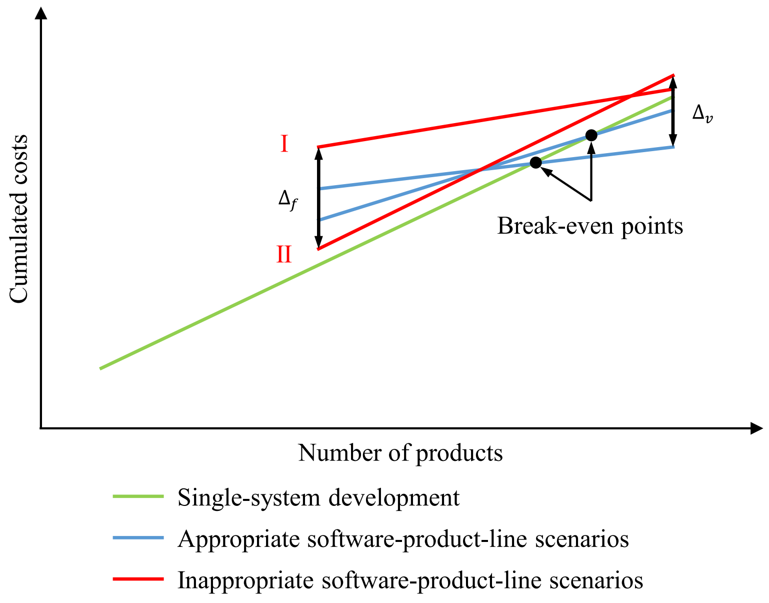
ii. A Software as a Service (SaaS) company started with a basic collaborative document editor. Over time, they've added optional features like chat, version control, and real-time co-editing. These are selectively enabled per customer.

iii. A national government is designing citizen's websites for different states. All websites must follow a strict federal specification with known features (e.g., tax filing, ID requests, appointments). No legacy software is reused.

iv. A small startup is planning a suite of finance apps: a budgeting tool, investment tracker, and tax calculator. All are being developed from scratch.

3. **Cost Analysis for the Extractive Adoption Strategy**

Explain the main message of the following figure:



In your answer refer to the following points:

(a) When is it beneficial to introduce a software product line with the extractive adoption strategy?

(b) How do the fix investment cost Δ_f relate with the variable cost Δ_v?

(c) Why are the scenarios I and II problematic?

(d) Refer and explain in your answer the terms *adoption barrier*, *return-on-investment* and *break-even-point*.

To answer this task, you may read the marked sections of the paper attached below:

Jacob Krüger, Wolfram Fenske, Jens Meinicke, Thomas Leich, and Gunter Saake. 2016. Extracting Software Product Lines: A Cost Estimation Perspective. In SPLC. ACM, NY, USA, 354–361.

4. **Choosing Implementation Techniques**

Which implementation technique(s) are most appropriate for the following product lines? Justify your answer based on the features and constraints provided.

(a) Mobile Banking App
Features: biometric login, multi-currency support, custom themes, in-app chat
Constraints: security-critical, cross-platform (Android/iOS), regulatory audits require feature traceability, only needed features should be delivered

(b) Game Engine with Mod Support
Features: physics engines, rendering backends, audio plugins, mod loader
Constraints: third-party mods must be integrated without recompilation, runtime loading of user-selected features, real-time performance must be preserved

(c) Smart Car
Features: real-time sensor monitoring, GPS, Lidar, cameras, voice UI, driver assistance, over-the-air updates
Constraints: deployed on varied hardware platforms, strict real-time constraints, must pass ISO 26262 safety certification, must run on low-memory devices (64 KB RAM)

(d) Mental Health App
Features: guided meditations, journaling, therapy chat, mood analytics
Constraints: each user has a different license profile, must support offline use, features are toggled on/off per user

Extracting Software Product Lines: A Cost Estimation Perspective

Jacob Krüger[1, 2]
jkrueger@hs-harz.de
jkrueger@ovgu.de

Wolfram Fenske[2]
wfenske@ovgu.de

Jens Meinicke[2]
meinicke@ovgu.de

Thomas Leich[1]
tleich@hs-harz.de

Gunter Saake[2]
saake@ovgu.de

[1]Hochschule Harz (FH) - University of Applied Sciences

[2]Otto-von-Guericke-University Magdeburg

ABSTRACT

Companies are often forced to customize their software products. Thus, a common practice is to clone and adapt existing systems to new customer requirements. With the extractive approach, those derived variants can be migrated into a software product line. However, changing to a new development process is risky and may result in unnecessary costs. Therefore, companies apply cost estimations to predict whether another development approach is beneficial. Existing cost models for software-product-line engineering focus on development from scratch. Contrarily, the extractive approach is more common in practice but specialized models are missing. Thus, in this work we focus on product-line extraction from a set of legacy systems. We *i)* describe according cost factors, *ii)* put them in context with the development process and cost curves, and *iii)* identify open challenges in product-line economics. This way, our work supports cost estimations for the extractive approach and provides a basis for further research.

CCS Concepts

•**Software and its engineering** → *Software product lines; Risk management;*

Keywords

software product line, extractive approach, cost estimation, investment analysis, risk assessment

1. INTRODUCTION

With increasing demand for customized software, companies are forced to develop multiple variants of the same system [23, 45, 51]. Software reuse enables an organization to derive systems from existing artefacts instead of developing them anew [35]. To manage and reuse similar products systematically, software product lines were adopted in software engineering [2, 36, 45]. Software product lines promise several benefits compared to single system development, such as, mass customization, reduced development and maintenance effort, and faster time to market [2, 13, 33, 45].

Despite such benefits, companies often apply single-system development because they fear costs and risks, face time limitations, or are unaware of suitable approaches [19, 32, 51, 54]. Instead, they apply unsystematic reuse, using the *clone-and-own* approach, to derive customized variants [2, 19, 23, 32]. This strategy describes cloning and modifiyng of a legacy system to adopt it to new customer needs [22, 52]. After cloning, a new and separated variant exists and can be assigned to developers [19, 50, 51, 56]. Summarized, clone-and-own is an opportunistic ad-hoc strategy to reuse software [13, 41, 52]. However, with the growth of variants, costs of development, maintenance, and customization increases [2, 50]. Due to such reasons, a company may consider migrating its cloned variants towards a product line. In practice this scenario is more common than developing a software product line from scratch [4, 20, 45, 51].

Developing new products is riskier than selecting successful variants that already exist. Thus, as legacy systems already exist, less market and risk analysis are required. Still, before a company migrates towards software-product-line engineering, it must estimate costs and benefits to justify the paradigm change [41]. There exist multiple cost models for product lines, such as the approaches of Poulin [47], SIMPLE [7, 8, 14], or COPLIMO [10]. However, re-engineering legacy systems is not considered in those models [34]. To address this gap, we make the following contributions:

- We analyse cost functions defined in SIMPLE and extend them with regard to migration of legacy systems. Thus, we identify additional cost factors, that must be considered in an according cost model.
- We put these cost factors in a systematic context with cost curves and the software-product-line development process. By doing this, we aim to show coherences of costs for software-product-line extraction.
- We identify and overview open challenges on cost estimations for the extractive approach. This way, we want to define a basis for further research in this area.

SPLC '16, September 16 - 23, 2016, Beijing, China

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4050-2/16/09...$15.00

DOI: http://dx.doi.org/10.1145/2934466.2962731

2. BACKGROUND

To describe cost estimations for the extractive approach, two topics are of interest. First, the extraction process for software product lines to identify specific costs. Second, the SIMPLE cost model that we refine for this extraction process.

2.1 Extracting Software Product Lines

There are essentially three approaches to develop software product lines [36]:

- Proactive - A product line is designed and implemented from scratch.
- Reactive - A small set or only one product is developed for systematic reuse. Later on, new features are added to extend the scope.
- Extractive - A product line is developed from a set of legacy systems. Common and variable parts of the existing implementations are re-engineered into reusable assets.

Proactive development is often considered to be the optimal strategy towards a product line [4, 12, 51]. In an empirical study, Berger et al. [4] asked for adoption strategies applied by the participating companies. 35.5% of the participants developed a product line proactive whereas the extractive approach was already used by 50%. One reason for this is that costs for extracting a product line is normally smaller than developing one from scratch [6, 12, 51]. Hence, proactive development is often not reasonable if cloned variants exist.

For the extractive approach, there are essential tasks during which costs occur [2, 36, 41]. First, commonalities and differences of legacy systems are analysed for feature identification. This can be supported by *feature location* techniques and tools [3, 18, 49]. After features and their dependencies are described, a *feature model* can be derived to describe variability of the product line [16]. Second, the identified features are re-engineered into reusable assets. During this task, it may be possible to reuse artefacts of the legacy systems. Otherwise, features must be implemented anew. Third, variability of assets, described by the feature model, must be implemented with a suitable technique. As result of these three steps, an extracted software product line exists.

2.2 SIMPLE

When switching to a new development approach, a company estimates the prospects of success. Cost models are a helpful means to predict efforts and savings. We base our considerations on the *Structured Intuitive Model for Product Line Economics* (SIMPLE) [7, 8, 14], a cost model for software product line engineering. It is not a calculation-based model but provides an overview and descriptions of relevant costs. For each function values can be determined with a cost estimation technique, such as, *judgement-based*, *algorithmic model*, or *analogies* [9].

A company's situation can be described with a general scenario [7]:

> "An organization has n product lines, each comprising a set of products, and $s1$ standalone products. It wants to have m product lines, each comprising a (perhaps different) set of products, and $s2$ stand-alone products. Along the way, the organization intends to add k products or delete d products."

From this definition additional scenarios can be derived. For example, Böckle et al. [7, 8] propose a scenario to excluded variants from an existing product line. Clements et al. [14] consider the migration of existing products towards a product line (the extractive approach). Depending on the scenario, coherences of the cost functions vary. Those functions are separated into *basic*, which describe the adoption process, and *evolution*, which consider the maintenance.

Basic Cost Functions.

SIMPLE defines four basic costs in software-product-line engineering:

1. C_{org} represents costs of introducing software-product-line development in an *organization*. Examples for cost drivers are: training, reorganization, and process improvement.
2. C_{cab} includes costs of building the *core asset base*. For instance, commonality and variability analysis, introduction of development environments, and architectural design, are considered.
3. C_{unique} describes efforts of implementing new requirements that are *unique* for a new product.
4. C_{reuse} represents costs of *reusing* features in a variant, for example, costs for identification, integration, and testing of assets.

Those cost functions are correlated as shown in Equation 1. The number of distinct products (p) that will be built is defined as n.

$$C_{SPL} = C_{org} + C_{cab} + \sum_{i=1}^{n}(C_{unique}(p_i) + C_{reuse}(p_i)) \quad (1)$$

To decide between product-line or stand-alone development, their costs are compared. Thus, SIMPLE defines the function $C_{prod}(p_i)$ which returns the costs to develop products without reuse. Savings for a software product line are estimated with Equation 2.

$$C_{savings} = \sum_{i=1}^{n} C_{prod}(p_i) - C_{SPL} \quad (2)$$

Finally, the return on investment (ROI) is calculated. To this end, the savings are compared to the required investments, as shown in Equation 3.

$$ROI = \frac{C_{savings}}{C_{org} + C_{cab}} \quad (3)$$

The basic cost functions only consider the adoption of software product lines. SIMPLE also includes life-cycle efforts, which we describe in the following.

Cost Function for Evolution.

In SIMPLE, maintenance costs are described as effort of releasing new versions (i.e., to fix bugs or extend the functionality). They are summarized as costs for evolution (C_{evo}), as illustrated in Equation 4. A new function, $C_{cabu}(p_i)$, is introduced to represent costs of *updating* the asset base. Changes to core assets can occur due to bug fixes or adaptations to a product. Such changes can have side effects on other variants that share commonalities. Thus, for all products the possibility of additional costs must be considered even if they remain unchanged. Efforts result from changes at the asset base, at product unique code, and for re-integration of changed assets into variants. For a cost estimation, maintenance costs for product-line engineering are compared to those of single-system development.

$$C_{evo} = \sum_{i=1}^{n} (C_{cabu}(p_i) + C_{unique}(p_i) + C_{reuse}(p_i)) \quad (4)$$

In conclusion, SIMPLE describes a set of cost functions. It enables companies to make a quick cost estimation for software-product-line development. However, SIMPLE only defines functions that support identification of costs but does not implement them. Thus, their calculation is left to the user. Additionally, SIMPLE is not adjusted accordingly to the extractive approach while it can still be applied.

3. COSTS OF EXTRACTING SOFTWARE PRODUCT LINES

In this section, we discuss the basis of cost estimations for the extractive approach. We first describe an according scenario and cost curves. Based on this, we analyse at which points SIMPLE's cost functions can be applied. For each of those cost functions, we discuss which additional characteristics should be taken into account for product-line extraction. Note, however, that a cost model incorporating these additional characteristics will be part of future work.

Cost estimations are applied in specific scenarios. For the extractive approach, we define the following scenario:

> A company owns a set of products that were developed via clone-and-own. To save costs, the organization considers switching its development towards software-product-line engineering. Thus, the company wants to estimate costs and savings of migrating its legacy systems.

Re-engineering legacy systems into a software product line requires additional efforts (*adoption barrier*). To evaluate whether this is useful, this extraction is compared to remaining cloning. Thus, if migration pays off, a *return-on-investments* will be achieved (*ROI*) after the *break-even-point* is reached. This pay-off is achieved either by reduced costs for development or maintenance, and often reported for three and more products [42, 45]. In Figure 1, we illustrate simplified cost curves for single-system and extractive product-line development to display this situation. We can see that extracting a number of products requires additional costs. Those costs are later compensated by reduced efforts for developing new variants (as we show in Figure 1) or maintaining the existing ones.

Based on this, we identify three questions a cost model for the extractive approach needs to answer:

Q-1 How much does extracting legacy systems into a software product line cost?

Q-2 How much does developing new products within the software product line cost?

Q-3 How much does software-product-line engineering save during maintenance per period compared to continuing clone-and-own development?

In the following sections, we discuss those questions and refine SIMPLE's cost functions. We describe costs that are specific for the extractive approach and analyse for which

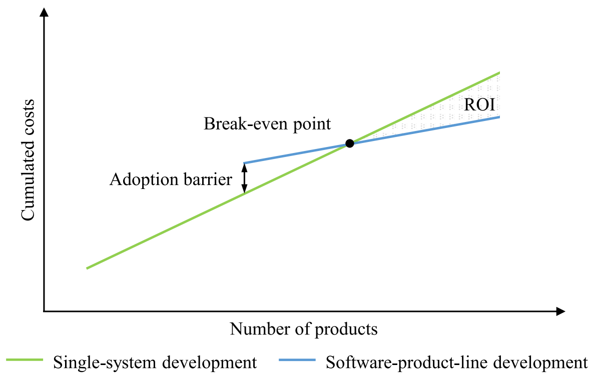


Figure 1: Simplified cost curves for extractive software-product-line development

product-line engineering tasks they are of interest. However, cost estimation involves uncertainty due to risks, such as market changes, and missing information [51]. We do not consider those uncertainties in this paper as they are unique for each company.

3.1 Organizational Costs

Organizational costs are represented in SIMPLE by the cost function C_{org}. Those costs are additional efforts that apply before a product line can be developed. Thus, we associate them with the first question (*Q-1*).

Contrarily to development from scratch, the extractive approach is based on legacy systems. To implement this approach, previously separated products and maybe developer teams must be merged. Thus, a first step for a company is to train participants and improve communication between them [6, 40, 43]. Furthermore, identical programming and documentation styles are necessary [46]. Therefore, a company has to unify, define, document, and follow structured workflows [6, 40, 43, 45]. In particular, processes and standards for reusing and merging legacy systems with different implementations must be defined. While the future reuse of artefacts can save efforts, this definition requires additional costs. In addition to those process refinements, a company may also require new tools [24]. In particular, tools for variability analysis and re-engineering are required to support migration processes.

To estimate organizational costs, a company must predict efforts of training employees and resulting benefits. Doing this precisely is difficult due to each person's individuality. According contextual information is complex and problematic to capture [26, 27, 29]. Thus, this task can hardly be supported with algorithmic cost models. Instead, estimations for SIMPLE's cost function must rely on judgement-based estimation and experience.

3.2 Costs of Extracting the Asset Base

In SIMPLE, extraction costs are represented with C_{cab}. They summarize all investments that are required to migrate legacy systems into an asset base. We associate those costs also with our first question (*Q-1*). By extracting functionalities of legacy systems into assets, a first software-prdouct-line instance is generated [2, 36, 41]. Based on the tasks necessary for the extractive approach (cf. Section 2.1), we identify three steps for a cost estimation:

1. Analyse legacy systems for possible features.
2. Estimate re-engineering effort and utility of features.
3. Decide which artefacts shall be migrated.

Following, we describe the three steps as well as their efforts and benefits in the extractive approach.

First, analysis of existing products identifies commonalities and differences among systems [20]. It provides information about possible features and their sizes. In contrast to proactive development, this process can be supported with feature location [18] or code-clone detection [48], which may require manual work [5, 30, 57]. In the extractive approach, this task benefits from experience of developers with the software. They are familiar with code, documentation, and design. Furthermore, analysis of the domain can be reduced as existing functionalities fulfil requirements of the corresponding market. The result of this step is the identification of feature candidates for extraction.

In the second step, efforts and utility of a feature's extraction are estimated. Re-engineering costs are influenced by size, dependencies, and the complexity of artefacts. Existing implementations with quality issues and outdated versions can be excluded. The predicted costs are compared to the utility of an asset. For example, a company can consider the number of products that contain the feature. This can be derived from the legacy systems. As a result, the company obtained an estimation about the costs for each feature candidate identified in the first step.

In the third step, the company has to decide which features it migrates. Based on the estimated utility and costs for the candidates, some of them are selected for extraction. For example, the company may only migrate the core features that are shared among all products. This reduces the investments compared to a full extraction as fewer migrations and variability implementations are necessary. However, more code clones remain and fewer variants can be instantiated. Therefore, maintenance and development costs may be higher. Thus, an organization can balance initial investment and required variability based on the artefacts it extracts. In conclusion, this steps results in a concrete set of features that shall be extracted.

In summary, the estimation of extraction costs supports a company's decision which artefacts are extracted. It provides essential information on the domain engineering and its costs. Those costs represent efforts in the domain application. An according cost model should be able to display different extraction scenarios and, thus, estimate costs for single assets.

3.3 Costs for New Products

We consider costs for new products within our second question (*Q-2*). SIMPLE separates costs to develop new code (C_{unique}) from costs to reuse assets (C_{reuse}). Still, both are required to describe the overall efforts for a new product. In this context, it is important to consider costs for a new variant with individual functions. Instantiating the same software frequently requires almost no effort [2]. Based on product requirements, the company identifies new features that must be developed. Those features can either remain stand-alone or are integrated into the asset base. In most cases, one additional asset provides a set of new variations. Due to this increase of possible configurations, it becomes difficult to manage variability [17]. In addition, assets are selected to instantiate the variant which results in costs for reuse. Still, the cost estimation is the same as for proactive development. However, information from legacy systems can support estimators, for instance with ratios of unique and reused code.

Due to those descriptions, we consider that costs for new products occur during two tasks:

- *Domain engineering* results in costs of developing new features. They do not remain stand-alone but are added to the asset base and, thus, belong to the domain.
- *Application engineering* results in costs for implementing unique code, reuse, configuration, and testing.

Thus, an algorithmic cost model should work on the level of individual features to differentiate between the development of unique product parts and core assets.

3.4 Maintenance Costs

Often, the main reason for a company to migrate its cloned product into a product line is to reduce maintenance costs [2]. Thus, estimating those costs is essential for the extractive approach. This is represented by our third question (*Q-3*) and summarized in the evolutionary cost function in SIMPLE (C_{evo}). To estimate maintenance savings, a company has to compare the respective efforts for single-system and software-product-line development. Those savings are determined for a number of products and a period of time to estimate when investments in a product line are compensated[10, 15, 37]. An organization may decide against migration if it only pays off after a long time. Some companies fear or cannot afford investments that do not provide savings early. Others may not plan for a long period or focus on keeping their products successful instead of introducing reuse [19].

An important point to address for the extractive approach is the quality of legacy systems. They may contain flaws in design and implementation. A company can remove those during adoption, but this requires additional investments. Alternatively, problems are fixed during maintenance. However, the implementation of variability increases the complexity of code and can result in new flaws [21] and removing errors becomes more expensive. Thus, a cost model for the extractive approach also needs to consider design flaws in particular. For example, they can be represented by cost drivers for additional tests or refactoring.

Despite its importance, only few existing cost models for product lines consider the life-cycle phase [1]. However, extracting a product line is often motivated by reduced maintenance costs. Thus, estimating those savings is essential for the extractive approach and must be considered in a suitable cost model.

3.5 Cost Estimation in the Software-Product-Line Development Process

This far, we described adaptations on cost estimations for the extractive approach. Following, we match those costs with the product-line-engineering process. We apply a nomenclature and clusters as we show in Figure 2. In the center (highlighted with grey background), we illustrate domain and application engineering based on Czarnecki and Eisenecker [16]. The other clusters illustrate the costs we described (without background colouring). As we show, the *organizational cost estimation* considers costs that occur before any development is done. The *domain cost estimation* cluster includes efforts for migration of legacy systems and

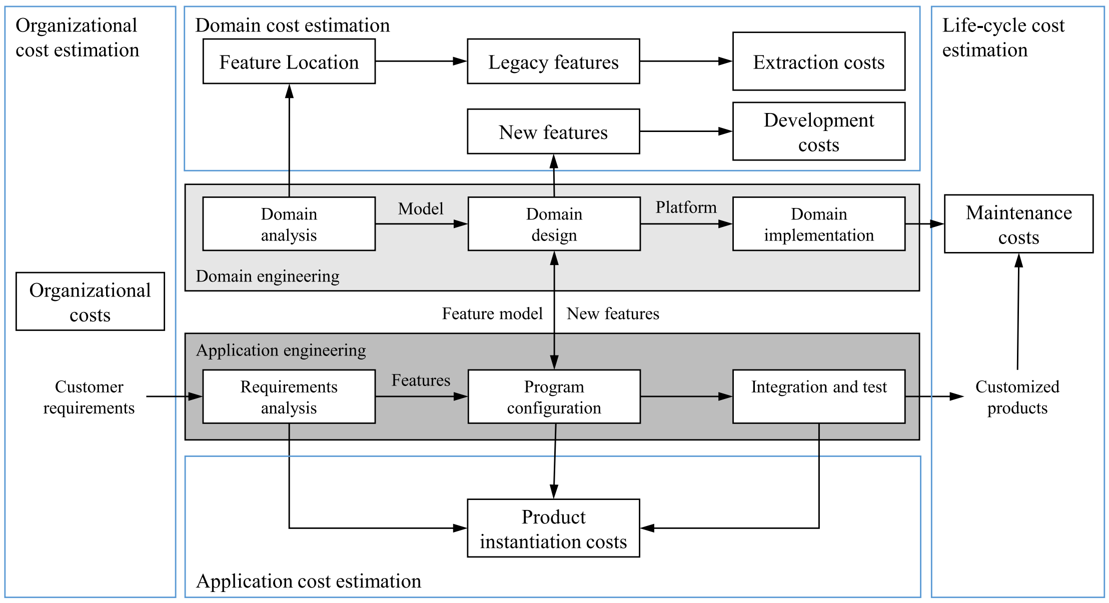


Figure 2: Cost estimation for the extractive approach, framed in blue. In the center, highlighted in grey, we show the product-line-engineering process adopted from Czarnecki and Eisenecker [16].

developing new assets. Extraction can be based on domain analyses and feature location while new features can be identified during the domain design. Those two clusters cover investments before a product line is usable. We consider costs for instantiating a product as *application cost estimation*. Those costs focus on configuring and testing a new variant. After the product line is extracted, maintenance costs occur. We summarize them in the *life-cycle cost estimation* cluster. Those costs consider the asset base and unique product parts. In conclusion, the described costs cover the software-product-line-engineering process. Thus, a cost model for the extractive approach must take them into account.

3.6 Cost Factors in the Cost Curve

As described in Section 3.2, a company specifies a set of assets it wants to extract from its legacy systems. This has not only impact on the investments but also on the resulting product line. In this section, we match this description with economic theory. Costs are often seperated into two categories [55]:

- *Fixed costs* (C_f) summarize all costs that are independent of the number of developed products. Thus, they remain constant during production. For example, fixed costs include investments for machinery or buildings. However, those costs are only fixed for a defined period. Additional investments might be necessary to increase production.
- *Variable costs* (C_v) cover the effort to develop a number of products. For instance, they include wages and costs for resources for each new product.

In Equation 5 we illustrate those basic costs (top) in comparison to SIMPLE's cost functions (bottom). We see that the costs for organizational activities (C_{org}) and asset base (C_{cab}) are not influenced by the number of developed products. Thus, they represent fixed, respectively adoption, costs to set up a software product line [45]. Efforts for reused (C_{reuse}) and unique (C_{unique}) parts depend on the number of developed variants. Thus, they belong to the variable costs.

$$\begin{aligned} C =& \quad \overbrace{C_{org} + C_{cab}}^{C_f} + \overbrace{\sum_{i=1}^{n}(C_{unique}(p_i) + C_{reuse}(p_i))}^{C_v * n} \\ C_{SPL} =& \end{aligned} \tag{5}$$

We see that fixed costs describe the adoption barrier, as shown in Figure 1. They represent the gap between retaining single systems and migrating towards a product line. The difference in variable costs is represented by the return-on-investment. However, organizations can vary their investments. Those have impact on efforts for development and during the life-cycle [9]. We consider fixed and variable costs as exchangeable economic goods. We can therefore display them as an *indifference curve* [44]. This means, that we can reduce investments at the expense of variable costs and vice versa. However, a full substitution is not possible. Consequently, companies face the following question: *Which is the optimal ratio between investment and resulting variable costs?* Due to uncertainties and unreliable data, predicting this ratio is problematic [51].

Another impact on the ratio has the selected implementation technique. Lightweight strategies require less investment than heavyweight ones but also increase the effort for new variants [42]. Thus, it is challenging to reliably define an investment that satisfies a specific quality. Instead, an organization may calculate several scenarios to consider uncertainties and migration strategies. For our example from Section 3.2, costs can be estimated for extracting all, or only some, legacy systems. In Figure 3, we show possible cost curves for extracted product lines with different investments (Δ_f). Due to the changing variable costs (Δ_v), the break-even points move. Some strategies are inappropriate for a company (red lines). For example, high investment costs (I) may require too many new products compared to other migration strategies. In contrast, low fixed costs (II) may never pay off because of insufficient benefits.

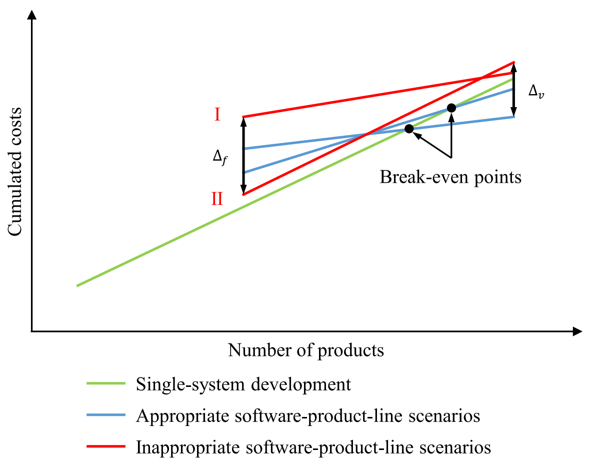


Figure 3: Possible scenarios for varying investments for the extractive approach

In this section, we described the basis of a cost model for the extractive approach. To do this, we discussed the cost function provided by SIMPLE. We refined those function to consider software-product-line extraction in more detail. In addition, we put them into context with product-line engineering. Finally, we connected the cost functions with economic theory and discussed coherence between them.

4. OPEN CHALLENGES

In this work, we described cost factors for the extractive software-product-line approach. There are several topics for cost estimation that are of interest and require further research.

4.1 A Cost Model for the Extractive Approach

Existing cost models for software product lines rarely consider legacy systems [34]. Thus, for the extractive approach companies must rely on expert knowledge. However, individual intuition, knowledge, and experiences can lead to varying results [26, 27, 29]. In contrast, algorithmic cost models provide a comprehensible and replicable result. They provide a structured way to predict efforts and can be repeated for several scenarios. Hence, an algorithmic cost model for the extractive approach is needed in practice. In current and further research we aim to develop such a model [38].

4.2 Data Extraction and Variability Mining

Variability mining provides semi-automatic tool support to analyze legacy systems [30]. Thus, it is possible to identify reliable data for cost estimations. Some examples for such data can be:

- *Source code comments* which can rate the documentation and, thus, understandability of code [53]. However, specialized tools for analyses and merging of multiple variants are required.
- *Re-usable* source code reduces migration efforts. For this purpose, numerous metrics exist [46]. Still, it is necessary to determine which can be helpful considering multiple legacy systems.
- *Code similarities* can help identifying feature candidates. In addition, predicting code size reductions is possible, which has the most impact on cost savings [10, 15]. However, a fully automatically approach to detect features and their sizes seems not realistic [5, 30]. Still, cross-product clone detection can provide clues [20, 58].

Identifying this data can provide helpful information not only for cost estimations but also the extraction itself. However, more research and specialized tool support is necessary.

4.3 Evaluation of SPL Cost Models

The evaluation of cost estimation approaches is a challenging task. In particular, there is a lack of reliable and suitable data [28, 31, 39, 43]. Often this can be explained as the publication of business data is critical for most organizations [34, 56]. Also, results gained from a single case study or with limited information cannot be generalized [1]. Evaluating cost estimations with experts is an alternative but in many cases their expertise and reliability is unclear [11, 27]. In the literature, several evaluation strategies are used but some approaches are not validated at all [1]. For example, Khurum et al. [31] compared 19 papers about software-product-line economics. Of those, only six were evaluated with case studies and eight used simplified or fictional data.

To overcome such problems, the authors propose a strategy for a systematic evaluation. At the beginning, an approach is applied in controlled experiments on replicable data. The results are discussed with experts and used for fine tuning. Afterwards, models are suitable for practical validation in case studies. While this is a first proposal for evaluation, the process requires more details and refinements. New evaluation methods, especially for comparing cost models, can provide further improvements.

5. RELATED WORK

In the context of our work, we focus on two related topics. First, economic descriptions of adoption strategies for product lines. Second, cost models for software product lines and their attributes.

Economic Comparison of Product Line Adoption.

Schmid and Verlage [51] compare the reactive and proactive adoption approaches for software product lines. They define different situations, in which these approaches are applied. The situations also take legacy systems into consideration. However, they do not focus on economic descriptions for those situations but provide an overview. By contrast, our work analyses costs factors of the extractive approach in detail.

Knauber et al. [33] define seven hypotheses on economic benefits of software product lines and quantify them. They describe savings that occur in general, such as reduced development efforts, but do not focus on a single adoption strategy. Contrarily, we only analyse specific costs for the extractive approach.

Clements and Krueger [12] discuss and compare benefits of approaches towards software-product-line adoption. They argue on advantages and disadvantages of the proactive and extractive approach. However, they do not focus on cost factors for product line adoption. This distinguishes their work from ours. We analyse the costs for the extractive approach in more detail.

Cost Models for Software Product Lines.
Ali et al. [1] categorize several cost models for software product lines. They provide a detailed overview on attributes and the cost factors. We also consider some of those attributes in our work, for example, consideration of the lifecycle and market analyses. However, their work only names these factors without analysing them in detail.

Heradio et al. [25] conducted a literature survey on the usage of feature models for cost estimations. They conclude, that considering the number of products which use a particular feature is essential to accurately estimate costs. We also emphasize the consideration of features instead of whole products. While we derive this argument from the extractive approach, they propose to use it to overcome simplifying assumptions.

6. CONCLUSIONS

In practice, companies often own a set of cloned legacy systems [2, 19, 23, 32]. To save costs they consider migrating those systems towards a product line. This is called the extractive approach to software-product-line adoption [36]. Still, this strategy is risky and requires investments. Cost models support companies in predicting chances of success. However, for the extractive approach a detailed and algorithmic cost model is still missing.

In this work, we discussed the basis for such an algorithmic cost model. To this end, we identified economic factors for software product lines with regard to the extractive approach. Considering legacy systems has not only impact on development efforts but also on cost estimation processes. Thus, we identified two main challenges. First, cost models must be adapted to the extractive scenario. Second, semi-automated analyses can provide more accurate input than estimating efforts based on expert judgement. Additionally, we matched our descriptions with product-line development and economic theory.

7. ACKNOWLEDGMENTS

This work is partly based on the first author's Master's thesis [38]. This research was partly supported by DFG grants LE 3382/2-1, SA 465/49-1, and BMBF grant 01IS14 017A.

References

[1] M. S. Ali, M. A. Babar, and K. Schmid. A Comparative Survey of Economic Models for Software Product Lines. In *SEAA*, pages 275–278. IEEE, 2009.

[2] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines.* Springer, 2013.

[3] W. K. G. Assunção and S. R. Vergilio. Feature Location for Software Product Line Migration: A Mapping Study. In *SPLC*, pages 52–59. ACM, 2014.

[4] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. A Survey of Variability Modeling in Industrial Practice. In *VaMoS*, pages 7:1–7:8. ACM, 2013.

[5] T. J. Biggerstaff, B. G. Mitbander, and D. Webster. The Concept Assignment Problem in Program Understanding. In *ICSE*, pages 482–498. IEEE, 1993.

[6] G. Böckle, J. B. Muñoz, P. Knauber, C. W. Krueger, J. C. S. do Prado Leite, F. J. van der Linden, L. M. Northrop, M. Stark, and D. M. Weiss. Adopting and Institutionalizing a Product Line Culture. In G. J. Chastek, editor, *SPLC*, pages 49–59. Springer, 2002.

[7] G. Böckle, P. C. Clements, J. D. McGregor, D. Muthig, and K. Schmid. Calculating ROI for Software Product Lines. *IEEE Softw.*, 21(3):23–32, 2004.

[8] G. Böckle, P. C. Clements, J. D. McGregor, D. Muthig, and K. Schmid. A Cost Model for Software Product Lines. In F. J. van der Linden, editor, *PFE*, pages 310–316. Springer, 2004.

[9] B. W. Boehm. Software Engineering Economics. *IEEE Trans. Softw. Eng.*, SE-10(1):4–21, 1984.

[10] B. W. Boehm, A. W. Brown, R. Madachy, and Y. Yang. A Software Product Line Life Cycle Cost Estimation Model. In *ISESE*, pages 156–164. IEEE, 2004.

[11] F. Bolger and G. Wright. Assessing the Quality of Expert Judgment: Issues and Analysis. *Decis. Support Syst.*, 11:1–24, 1994.

[12] P. C. Clements and C. W. Krueger. Point / Counterpoint: Being Proactive Pays Off / Eliminating the Adoption Barrier. *IEEE Softw.*, 19(4):28–31, 2002.

[13] P. C. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2006.

[14] P. C. Clements, J. D. McGregor, and S. G. Cohen. The Structured Intuitive Model for Product Line Economics (SIMPLE). Technical Report CMU/SEI-2005-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[15] S. G. Cohen. Predicting When Product Line Investment Pays. Technical Report CMU/SEI-2003-TN-017, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.

[16] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2005.

[17] S. Deelstra, M. Sinnema, and J. Bosch. Product Derivation in Software Product Families: A Case Study. *J. Syst. Software*, 74(2):173–194, 2005.

[18] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature Location in Source Code: A Taxonomy and Survey. *J. Softw. Evol. and Proc.*, 25(1):53–95, 2013.

[19] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. An Exploratory Study of Cloning in Industrial Software Product Lines. In *CSMR*, pages 25–34. IEEE, 2013.

[20] S. Duszynski, J. Knodel, and M. Becker. Analyzing the Source Code of Multiple Software Variants for Reuse Potential. In *WCRE*, pages 303–307. IEEE, 2011.

[21] W. Fenske and S. Schulze. Code Smells Revisited: A Variability Perspective. In *VaMoS*, pages 3–10. ACM, 2015.

[22] W. Fenske, T. Thüm, and G. Saake. A Taxonomy of Software Product Line Reengineering. In *VaMoS*, pages 4:1–4:8. ACM, 2014.

[23] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In *ICSME*, pages 391–400. IEEE, 2014.

[24] P. Gacek, Cristina amd Knauber, K. Schmid, and P. C. Clements. Successful Software Product Line Development in a Small Organization: A Case Study. Technical Report 013.01/E, Fraunhofer IESE, Kaiserslautern, Germany, 2001.

[25] R. Heradio, D. Fernandez-Amoros, J. A. Cerrada, and I. Abad. A Literature Review on Feature Diagram Product Counting and its Usage in Software Product Line Economic Models. *Int. J. Soft. Eng. Knowl. Eng.*, 23(8):1177–1204, 2013.

[26] M. Jørgensen. A Review of Studies on Expert Estimation of Software Development Effort. *J. Syst. Software*, 70 (1):37–60, 2004.

[27] M. Jørgensen. Forecasting of Software Development Work Effort: Evidence on Expert Judgement and Formal Models. *Int. J. Forecast.*, 23(3):449–462, 2007.

[28] M. Jørgensen and M. Shepperd. A Systematic Review of Software Development Cost Estimation Studies. *IEEE Trans. Softw. Eng.*, 33(1):33–53, 2007.

[29] M. Jørgensen, B. W. Boehm, and S. Rifkin. Software Development Effort Estimation: Formal Models or Expert Judgment? *IEEE Softw.*, 26(2):14–19, 2009.

[30] C. Kästner, A. Dreiling, and K. Ostermann. Variability Mining: Consistent Semi-Automatic Detection of Product-Line Features. *IEEE Trans. Softw. Eng.*, 40(1): 67–82, 2014.

[31] M. Khurum, T. Gorschek, and K. Pettersson. Systematic Review of Papers About Economic Solutions for Product Lines. In *MESPUL*, pages 277–284. IEEE, 2008.

[32] P. Knauber, D. Muthig, K. Schmid, and T. Widen. Applying Product Line Concepts in Small and Medium-Sized Companies. *IEEE Softw.*, 17(5):88–95, 2000.

[33] P. Knauber, J. Bermejo, G. Böckle, J. C. S. do Prado Leite, F. J. van der Linden, L. M. Northrop, M. Stark, and D. M. Weiss. Quantifying Product Line Benefits. In F. J. van der Linden, editor, *PFE*, pages 155–163. Springer, 2002.

[34] H. Koziolek, T. Goldschmidt, T. de Gooijer, D. Domis, S. Sehestedt, T. Gamer, and M. Aleksy. Assessing Software Product Line Potential: An Exploratory Industrial Case Study. *Empir. Software Eng.*, pages 1–38, 2015.

[35] C. W. Krueger. Software Reuse. *ACM Comput. Surv.*, 24(2):131–183, 1992.

[36] C. W. Krueger. Easing the Transition to Software Mass Customization. In F. J. van der Linden, editor, *PFE*, pages 282–293. Springer, 2002.

[37] C. W. Krueger. Towards a Taxonomy for Software Product Lines. In F. J. van der Linden, editor, *PFE*, pages 323–331. Springer, 2004.

[38] J. Krüger. A Cost Estimation Model for the Extratcive Software-Product-Line Approach. Master's thesis, Otto-von-Guericke-Univerity Magdeburg, 2016.

[39] H. K. N. Leung and Z. Fan. Software Cost Estimation. In *Handbook of Software Engineering and Knowledge Engineering*, pages 307–324. World Scientific Publishing, 2002.

[40] J. Mansell. Experiences and Expectations Regarding the Introduction of Systematic Reuse in Small- and Medium-Sized Companies. In T. Käköla and J. Duenas, editors, *Software Product Lines*, pages 91–124. Springer, 2006.

[41] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. Le Traon. Bottom-Up Adoption of Software Product Lines: A Generic and Extensible Approach. In *SPLC*, pages 101–110. ACM, 2015.

[42] J. D. McGregor, L. M. Northrop, S. Jarrad, and K. Pohl. Guest editors' introduction: Initiating software product lines. *IEEE Softw.*, 27(3):16–21, 2002.

[43] L. M. Northrop. SEI's Software Product Line Tenets. *IEEE Softw.*, 19(4):32–40, 2002.

[44] V. Pareto. *Manuale di Economia Politica.* Societa Editrice, 1906.

[45] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005.

[46] J. S. Poulin. Measuring Software Reusability. In *ICSR*, pages 126–138. IEEE, 1994.

[47] J. S. Poulin. The Economics of Software Product Lines. *Int. J. Appl. Softw. Technol.*, 3(1):20–34, 1997.

[48] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Sci. Comput. Program.*, 74(7):470–495, 2009.

[49] J. Rubin and M. Chechik. A Survey of Feature Location Techniques. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. G. Cohen, and J. Bettin, editors, *Domain Engineering*, pages 29–58. Springer, 2013.

[50] J. Rubin, A. Kirshin, G. Botterweck, and M. Chechik. Managing Forked Product Variants. In *SPLC*, pages 156–160. ACM, 2012.

[51] K. Schmid and M. Verlage. The Economic Impact of Product Line Adoption and Evolution. *IEEE Softw.*, 19 (4):50–57, 2002.

[52] S. Stanciulescu, S. Schulze, and A. Wasowski. Forked and Integrated Variants in an Open-Source Firmware Project. In *ICSME*, pages 151–160. IEEE, 2015.

[53] D. Steidl, B. Hummel, and E. Juergens. Quality Analysis of Source Code Comments. In *ICPC*, pages 83–92. IEEE, 2013.

[54] A. Tang, W. Couwenberg, E. Scheppink, N. A. de Burgh, S. Deelstra, and H. van Vliet. SPL Migration Tensions: An Industry Experience. In *KOPLE*, pages 1–6. ACM, 2010.

[55] J. Viner. Cost Curves and Supply Curves. *Zeitschrift für Nationalökonomie*, 3(1):23–46, 1932.

[56] K. Yoshimura, D. Ganesan, and D. Muthig. Defining a Strategy to Introduce a Software Product Line Using Existing Embedded Systems. In *EMSOFT*, pages 63–72. ACM, 2006.

[57] K. Yoshimura, D. Ganesan, and D. Muthig. Assessing Merge Potential of Existing Engine Control Systems into a Product Line. In *SEAS*, pages 61–67. ACM, 2006.

[58] T. Ziadi, L. Frias, M. A. A. d. Silva, and M. Ziane. Feature Identification from the Source Code of Product Variants. In *CSMR*, pages 417–422. IEEE, 2012.