Predicting and Learning the Performance of Configurable Software Systems

Lion Wagner

University of Stuttgart
Institute of Software Technology (ISTE)
70569 Stuttgart, Germany

Abstract. This is my abstract.

1 Introduction

Todays programs are mostly highly customizable. Whilst buying or downloading a program a user already decides on which version of program (s)he wants to have. On Installation there are mostly multiple Options reaching from 'Language' to a selection of software features. Once a program is installed there are usually multiple startup, customization and configuration options provided for a customer. But having such a large amount of options brings some problems into the development of modern applications. We need to have a stable development strategy that can offer the creation of multiple similar products with little redundant code or components.In practice software product lines are used for this case

Furthermore there is the problem of performance prediction. For this we introduce some Definitions. This paper will use the definition of *feature* and *configuration* as they are described in [5] "[...], where a feature is a stakeholder-visible behavior or characteristic of a program." and "a specific set of features, [is] called a configuration". The amount of possible configurations naturally lies in $\mathcal{O}(2^n)$. This scaling makes it hard to test each singular configuration for its performance or correctness. Especially if the configuration under test is unpredictably chosen by a user [5]. Nevertheless this paper will only look at the non-functional performance properties of an application. Consequently efficiently analyzing a product's performance is only partially possible and behavior beyond that has to be predicted. This opens up the challenge of efficiently and accurately predicting performance. Over time a lot of methods in different disciplines have been prosed.

This paper aims to give a short introduction into the importance of software product lines (SPL) in regards to performance engineering and an overview over solutions for predicting and learning the performance of a configurable software system. mention references   TODO

2 Software Product Lines Performance Influence

Intro tarditionell präprözessoren oder compiler optionen oder startup optionen

2.1 Introduction into Software Product Lines

As mentioned above modern software systems are often highly configurable and customizable. To be able to provide such a large amount of customization, software engineers adopted the concept of *mass customization* inside a product line. A technique mostly known from the automotive industry[3].[3] further describes how a general product-line functions: There is a *common platform*, which is in itself expandable and holds all basic functionalities. It also provides interfaces for adding on parts (features). Some parts are mandatory and some are optional. For each part there might be different versions or variations that are interchangeable.
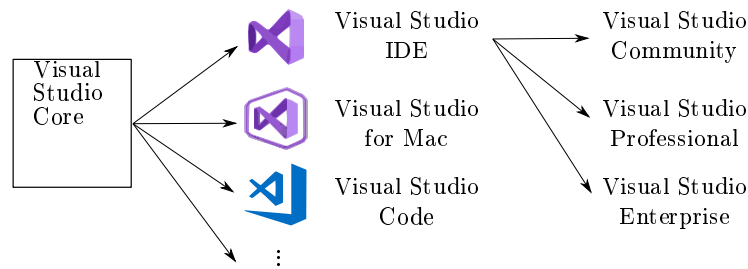
Fig. 1: Partial view of the Visual Studio Product Line [1]. (Assuming a Microsoft is using core libaries for its product. Which is likely since .Net Core runs on all major operating systems.)

Software product lines (SPL) inherit this behavior of reusing existing components to enhance the development process [2]. They are used to create a software product that is highly configurable and customizable. Products inside a SPL can even serve as a common platform for further customization. A more detailed look at the economical thought behind SPLs is given by Díaz-herrera et al. [2]. An example of a SPL would be the Visual Studio product line from Microsoft as shown in figure 1. The common platform are some core libraries that provide shared code to all products of this line. The products themselves add platform or usage specific features (parts) to the core to offer a broad variation of products. The Visual Studio IDE even splits up further into 3 products that all serve a different purpose, but use same code base. Note that figure 1 only displays static variations of the product line. Almost all versions are additionally dynamically expendable via plug-ins and customizable via options and preferences.

2.2 Performance Influences of Feature Binding in SPLs

The following section is based on Combining Static and Dynamic Feature Binding in Software Product Lines by Rosenmüller et al. [4].

When designing a SPL one has a choice for each feature to either bind it statically or dynamically. A statically bound feature is compiled directly into the binary of a program. Its always available to the environment when needed but uses up binary space to do so. When dynamically binding a feature it is not a part of the core binary, yet a part of an external *feature library*. Once such a feature is needed the core loads the according library from an external source. This leads to the core binary being smaller but the size needed for each feature is increased by some meta-data (size, location, format, etc. of the feature library). Multiple features can be combined into one feature library for optimization. Feature libraries are also called *binding units*. Note that [4] uses the *decorator pattern* for the realization of dynamic feature binding.

To describe the effects of binding types we use the terms *functional* and *compositional overhead*. A functional overhead originates form features (or binding units) that are loaded inside a program yet never used. In turn a compositional overhead is found as meta data that is needed to describe dynamic behavior.

An abstract example: A program core consisting out of 3 features and uses up 2MB of (hard-drive) memory. We extract 1 feature that is rarely used into a dynamic link library with the size of 1.1MB. Afterwards our core is only 1MB large. We can already conclude that our original program had a functional overhead of 1MB by subtracting the old and the new binary size. When using the extracted feature our program uses up 2.1MB of memory. So we can say that our compositional overhead for extracting the feature is 0.1MB. That is the data needed to describe the location, format, etc. of the feature library.

daten aus paper einfügen   TODO

Fig 18 einbauen Coming back to binding types we have the problem of find-   TODO
ing an optimal combination of static and dynamic bindings. Rosenmüller et al. [4] suggest to prefer static binding if no dynamic extensibility is required and resources are not limited. A full static bound program has no compositional overhead and thus only requires additional (persistent and transient) memory. However programs that are more limited in their resources may need to be designed to minimize the overall overhead. This is basically the same task of finding an optimal configuration of a program. Techniques regarding the analysis and prediction will come up later in the paper reference einfügen. The compositional   TODO
overhead of binding units can be predicted partially since feature libaries have a constant size header (e.g. 5KB per Dynamic Link Library {DLL}). A general guideline is to avoid a large number and a large size of binding units. Overlapping binding units as well as dynamic splitting and merging of binding units are possible optimizations. Rosenmüller et al. [4] also mention that "[...] there is no optimal size for a binding unit, a domain expert can define binding units per application scenario.".

3 Measuring and Predicting the Performance of Highly Configurable Systems

With such a large amount of customization options a

4 Measuring Option and Problems

5 Prediction

References

[1] `https://visualstudio.microsoft.com/de/products/`. Accessed: 10.05.2019.

[2] J. L. Díaz-herrera, P. Knauber, and G. Succi. Issues and models in software product lines. 2000.

[3] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag Berlin Heidelberg, Tiergartenstraße 17, 69121 Heidelberg, 2005.

[4] M. Rosenmüller, N. Siegmund, G. Saake, and S. Apel. Combining static and dynamic feature binding in software product lines. 2009.

[5] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting performance via automated feature-interaction detection. In *In Proc. of ICSE*, pages 167–177. IEEE, 2012.