Reuse of automotive Software – Maturity Model, Technology and Report from Praxis

Dr. Thomas Zurawka, SYSTECS Informationssysteme GmbH

1 Introduction

Today's vehicles of all kinds are driven by a lot of computers, called **E**lectronic **C**ontrol **U**nits (ECU). These ECUs contain a huge and steadily increasing amount of embedded **Software**.

1.1 ECU Software Architecture and V-Cycle Phases

The high level software architecture of today's ECUs is shown in figure 1-1. The ECU software contains at least two layers: Application software and Basic software [1]. Each layer consists of software components (**SWC**). A SWC consists of Code and Data.

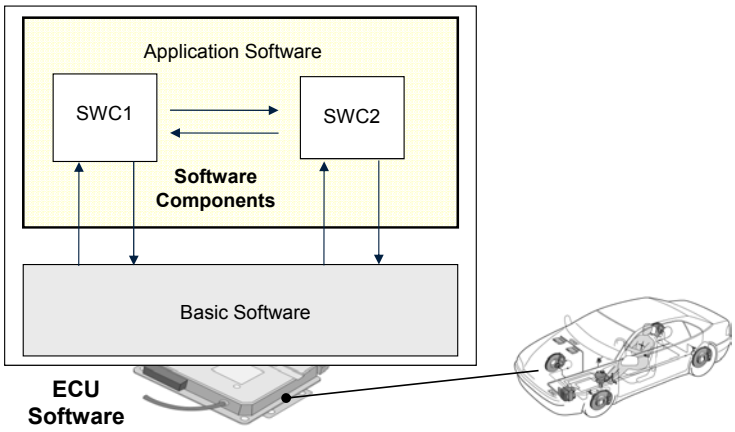

Figure 1-1: High Level Architecture of ECU Software

The ECU software and their SWC will be developed by means of certain development steps within the so called "V-Cycle" [1]. This is shown in more detail in figure 1-2.

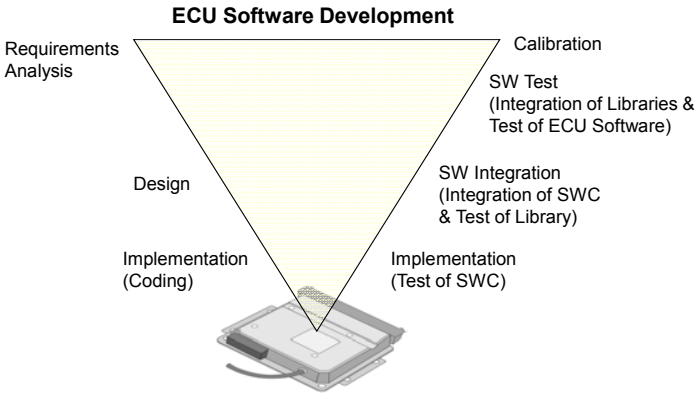

Figure 1-2: Development of ECU Software ("V-Cycle")

Reduction of development costs and increase of software quality are key objectives of all automotive software development organizations.
The **reuse of SWC** and other artefacts of the software development process is a widely accepted key measure in order to achieve these goals [2].

Therefore SYSTECS, together with customers, has developed a new maturity model for the reuse of software in order to allow organizations to migrate step-by-step to a higher level of reuse over the years. This migration path including the target maturity level must be planned very carefully in order to actually reduce the development costs.

2.1 Some Definitions

Planned reuse is the intended design and implementation of SWC in order to reuse them later [6]. For simplification reasons the term SWC will be used from now on for all reusable artifacts.

Product: piece of Software developed by an organization, e.g. ECU software

Application Domain: environment for which the products will developed for. E.g. ECU software for the domain “control of gasoline engines”

Product Context: refinement of the application domain. This may include e.g. for the product ECU software:
- e.g. types of engines or transmissions
- ECU hardware, e.g. memory size
- Sensors and Actuators
- Design principle of control algorithms
- Development environment

Feature: Variant Management includes the management of variability within a product. The description of the variability will be done by means of so called **Features**. A feature contains several requirements which must be bundled together from a customer point of view.

Type of reuse:
- Versions of SWC: widely used and very well understood
- Variants of SWC: widely used and very well understood
- Intra product oriented reuse: widely used and well understood (e.g. arithmetic libraries)
- Inter product oriented reuse: less used and less understood

Layer of reuse:
- Product requirements analysis
- Product design
- Product implementation (mostly covered if we talk about reuse)
- Product integration
- Product test
- Product calibration

Application of reuse:
- Vertical reuse within the same application domain (e.g. diesel engine software)
- Horizontal reuse across application domains (e.g. across gasoline engine & transmission ECU software)

White box reuse (Clone & Own)
- Existing SWC will be modified, adapted and tested additionally
- Existing SWC will not be stored in reuse archive
- Versions or copied and original SWC are different
- Maintenance and bug fixing are done many times

White box reuse is widely introduced.

Black box reuse:
- Existing SWC will not be changed in case of reuse for other products. A configurable SWC must be configured before reuse. It must be ensured that the configuration itself is reproducible.

Black box reuse offers by far the biggest potential for reduction in software development costs and increase in quality [2].
Horizontal reuse of SWC is a very challenging issue in the automotive industry. This objective is currently driven mainly by the AUTOSAR initiative which will finally lead to a horizontal reuse of the just the basic software across all organizations and application domains.

The objective is therefore to introduce planned, inter product oriented and vertical black box reuse of SWC on all layers.

2 Maturity Model

The following maturity model for reuse of software has been used successfully by SYSTECS in different consulting projects for ECU software suppliers (figure 2-1).

The higher the maturity levels the higher the degree of black box reuse of SWC. The higher the reuse of SWC the higher the software quality and the lower the software development cost. The exact relationship between these two issues is nonlinear and will be refined later in more detail.

2.1 Introduction

Level 0 – No Reuse:
Completely new development of software for each product

Level 1 – ad-hoc Reuse:
White-Box Reuse of software only (“Clone & own”). No systematic, no coordinated and no documented reuse of software.

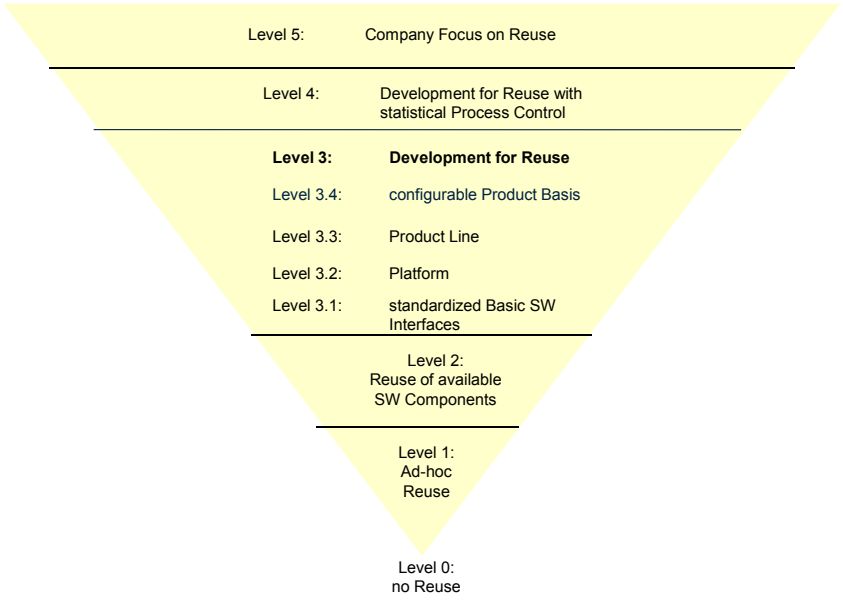

Figure 2-1: Maturity Levels for Reuse of Software

Level 2 - Reuse of available software components:
Collection, structuring and documentation of available (company internal and company external) SWC in a systematic way.

Level 3 – Development for Reuse

Level 3.1 – standardized Basic Software Interfaces:
There are stable interfaces between application software and basic software. The new automotive standard AUTOSAR will standardize these interfaces in the near future. This level will typically be achieved if some experience with the application domain exists.

Level 3.2 – Software Platform:
A software platform consists of common SWC which will be used by **all** products. Existing software architecture is mandatory for this level. This level will typically be achieved for mature application domains for which the common features of all products have been already identified.

Level 3.3 – Software Product Line
A software product line consists of SWC which will be used by some of the products. These SWC must therefore not be integrated into a software platform. Existing software architecture is mandatory for this level. This level will typically be achieved for mature or established application domains for which the **common and different** features of all products have been already identified. Additionally the software development organization must have deep knowledge of the application domain.

Level 3.4 – configurable Product Basis
A configurable product basis is an advanced software product line. All products will automatically be build by existing SWC of a product line. There are no customer specific SWC. Each SWC contains a lot of features. This level can only be achieved for very established application domains. Additionally the software development organization must have deep knowledge of the application domain and long-term experience with software engineering in order to handle the huge complexity.

Level 4 – Development for Reuse with statistical process control:
This level includes level 3 capabilities including statistical process control in order to permanently supervise and improve the degree of reuse of SWC.

Level 5 – Company focus on Reuse:
The whole company, especially sales and marketing, is focusing on reuse of SWC.

2.2 Some practicable Hints

The maturity enables a **rating** of exactly the software development organization which will be required to develop the products. The term development includes all phases of software engineering, starting from requirements analysis to the release of the software product. Within automotive software engineering this includes calibration of software as well since data is part of software.
The maturity model will in general directly be applied to the following issues:

1. Products which will be developed
2. Context of these products
3. Rated Organization: whole organization which will be required to develop the product

Example 1:
Product: ECU Software (code and data!) for control of diesel engines
Context: Passenger Car diesel engines, Memory size < 4 MByte Flash/EEPROM.
Rated Organization: Departments X, Y of company Z.

The maturity model can also be applied to a simple SWC like an adder (with different features) which will be developed by a single software developer.

Example 2:
Product: Adder
Context: arithmetic fixed point library
Rated Organization: software developer X.

A more precise rating is possible if we analyze each development phase and each SWC separately (figure 2-2 and 2-3).

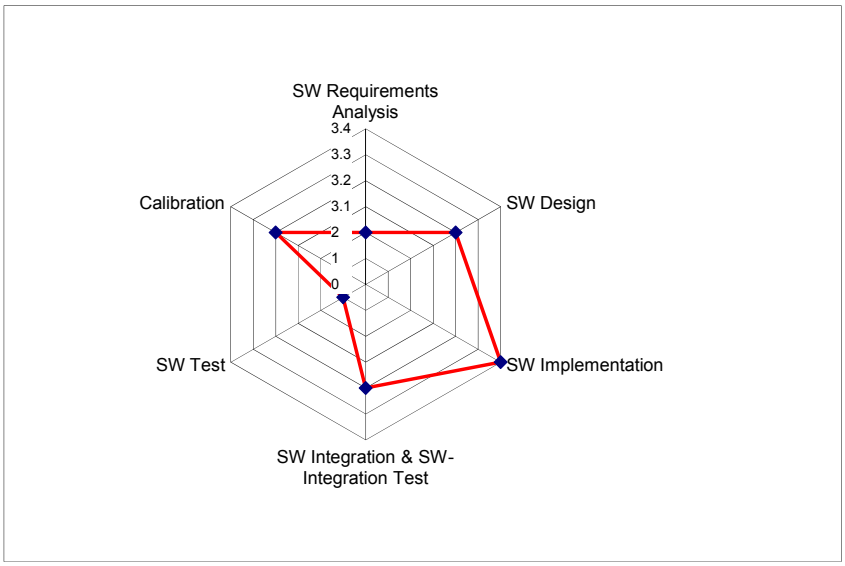

Figure 2-2: Maturity Levels for Reuse of Software (V-Cycle View)

Since **code variants** as well as **data variants** are widely used within automotive software engineering the maturity model cover both issues. Code variants are used from software requirements analysis to software test and data variants are used within the calibration phase.
A high maturity level for calibration can only be achieved if there exist a high level for the other phases as well since reuse of data is almost not possible if there is no reuse of code.

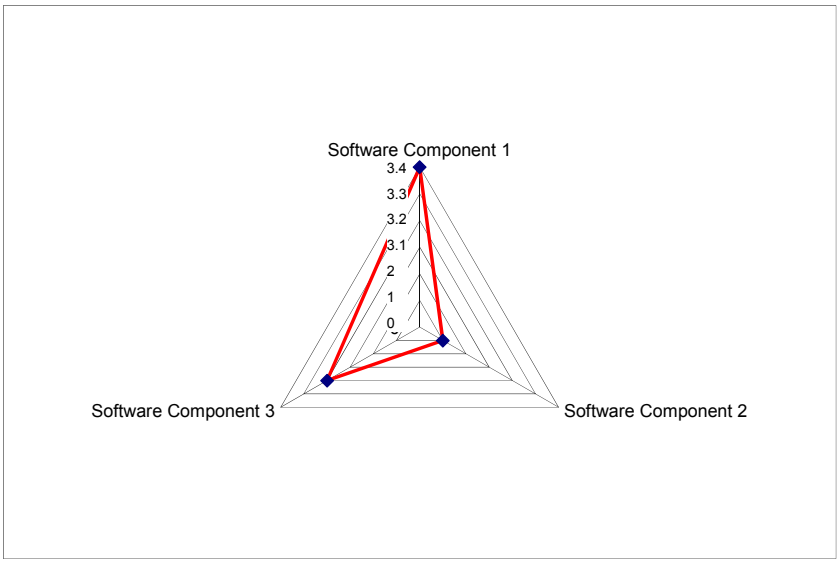

Figure 2-3: Maturity Levels for Reuse of Software (SWC View)

3 Organizational Aspects of Reuse

Today reuse of SWC is not primarily a technical challenge it's **mainly an organizational challenge**. A software development organization has to introduce a **culture for reuse**, e.g. by introduction of the following measures [2]:

- SWC oriented development organization instead of a pure project oriented organization
- Development process which support reuse
- Steering committee for balancing time pressure within projects and reuse of SWC
- Benefits for providers and users of reusable SWC
- Reuse archive with 2 parts: (1) tested SWC, (2) provided, but not yet completed and tested SWC. Existing SWC must be found very fast and convenient ("Google")
- (less) Metrics in order to control the progress

The introduction of a product line is an innovative process and must be handled as such [2]. It takes **at least 2 years** to move to a higher maturity level. This must be done step by step.
The following rules of thumb [6] may help:

- A SWC must be developed **3 times** before it can be reused
- Cost/Benefit < 1 if a SWC has been reused **3 times**

The organizational aspects are usually extremely important. The above mentioned issues, if carefully considered, will help to implement a product line over time.

4 Technological Aspects of Reuse

The technological aspects for the maturity levels 0 to 3.1 are well known and simple [2]. The technology for maturity levels higher than 3.2 (platform) is more sophisticated and must include the following issues [3,4,5]:

- Product & Product Context
- Software Architecture
- Feature Modeling and Configuration
- Scoping
- Management of Variants for
    - Software Requirements Analysis
    - Software Design
    - Software Implernentation
    - Software Integration
    - Software Test
    - Calibration

The definition of the product, e.g. ECU software, and the product context is a very important element since the ECU software must finally run in this environment. A platform or a product line will be designed for a certain context!

The definition of a software architecture is well understood and explained in [2].

Scoping is the identification of common (mandatory & variant) and special features of all products of an application domain. It solves the question: Which product contains what features?
Figure 4-1 shows this in more detail.

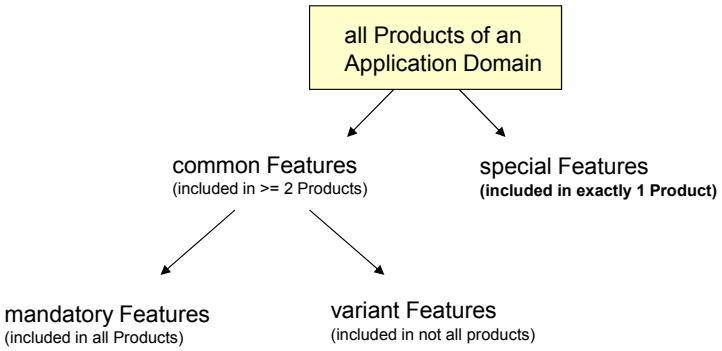

Figure 4-1: Scoping

Example: for very mature application domains like “ECU software for passenger cars with gasoline engines” the degree of common features is very high. For emerging application domains like “ECU Software for Hybrid Vehicles” the degree of common features is low since many new ideas of different customers will result in special features.

Scoping is of crucial importance for the decision which target maturity level should be chosen for a certain organization.

Feature modeling includes the definition of all features of all products of an application domain and their dependencies [4]. Feature configuration is the selection of features for a certain product [4]. Figures 4-2 shows this in more detail.
Finally all phases of the V-Cycle must support the feature mechanism. This issues is itself rather sophisticated since it depends heavily on the chosen development methods and tools. E.g. for software implementation most often C/C++ preprocessor macros are used in order to implement different features. More information can be found in [2,3,4,5].

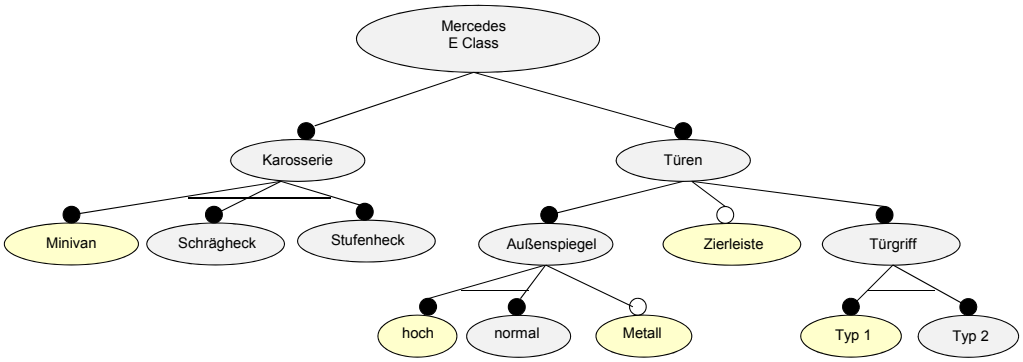

Figure 4-2: Feature Modeling and Configuration

The handling of all the features differs now significantly for each maturity model. Figure 4-3 to 4-5 shows this in more detail.

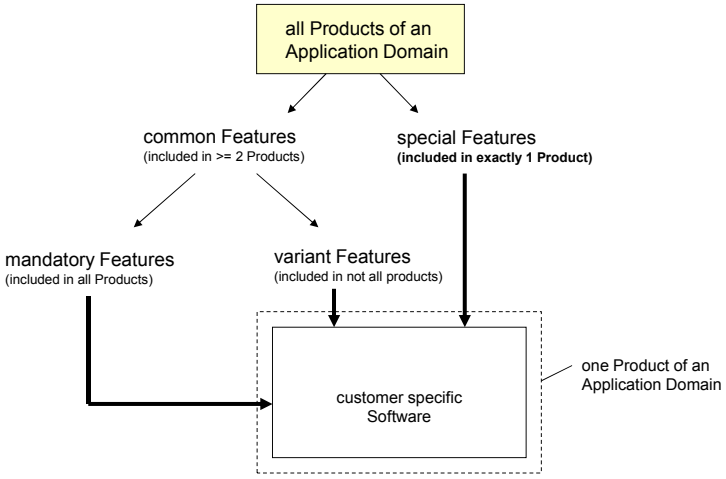

Figure 4-3: Feature assignment for Maturity Level 0 to 3.1

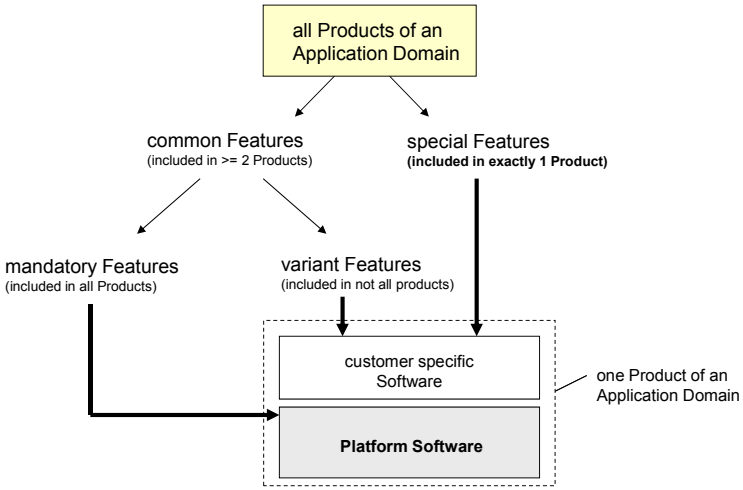

Figure 4-4: Feature assignment for Maturity Level 3.2 (Platform)

It is important to mention that within platform software the different features are not implemented in a configurable way. This is the difference to a product line where each feature can be configured separately.

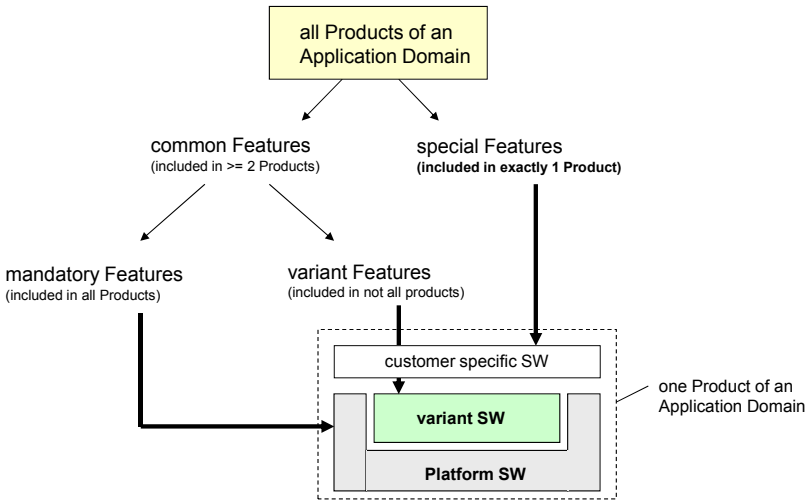

Figure 4-5: Feature assignment for Maturity Level 3.3 (Product Line)

Figure 4-3 to 4-5 shows directly that the higher the maturity level the higher the reuse of features and therefore of software. Nevertheless the decision for the right target maturity level is a sophisticated one since it depends on the spilt of mandatory, variant and special features. This is the reason why the scoping phase is so important. The next chapter will show this in more detail.

5 SW Development Costs & Target Maturity Level

The following is a summary of SYSTECS experience from several consulting projects.

5.1 Normalized Software Development Costs

In general the total software development costs (SWDC) for an organization which is developing ECU software by means of many projects depends on several issues. The following equation is a summary of simple math (see also [2]) and SYSTECS long term experience in automotive software engineering:

SWDC = f (LOC, EL, CO, N, R)

LOC	Lines of Code for an ECU
EL	ideal effort for 1 LOC (without complexity increase due to reuse technology) EL = f (Performance of an employee, efficiency of processes, methods & tools)
CO	Cost of a person year; CO= f (Salary of employee, infrastructure)
N	Number of projects (cumulated)
R	= f (a, b, c, N); normalized SWDC per project a – Degree of black box reused SWC b – Normalized SWDC for white box reused SWC b= f (Degree of change of a SWC during White box reuse) c – Increase of complexity due to reuse (Typical: effort per line grows linear with a; e.g. c=10%*a)

Typically LOC, CO and N are well known for a given ECU software development organization. The factor EL depends heavily on the employees and their working environment. All these factors are **independent** on the degree of reuse of black box SWC.

The normalized SWDC per project R depends on the degree of black box reuse of SWC and on the normalized SWDC for white box reused SWC (the ones which cannot be black box reused). Figure 5-1 (based on [6]) and 5-2 show the relationship in more detail.

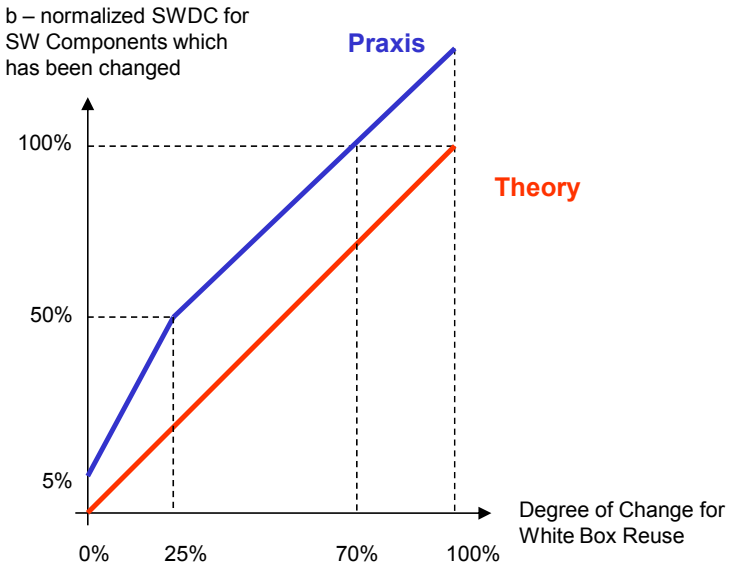

Figure 5-1: Normalized SWDC for white box reused SWC

Figure 5-1 shows that if we change 70% of a SWC the effort is equal to the original effort to develop these SWC.

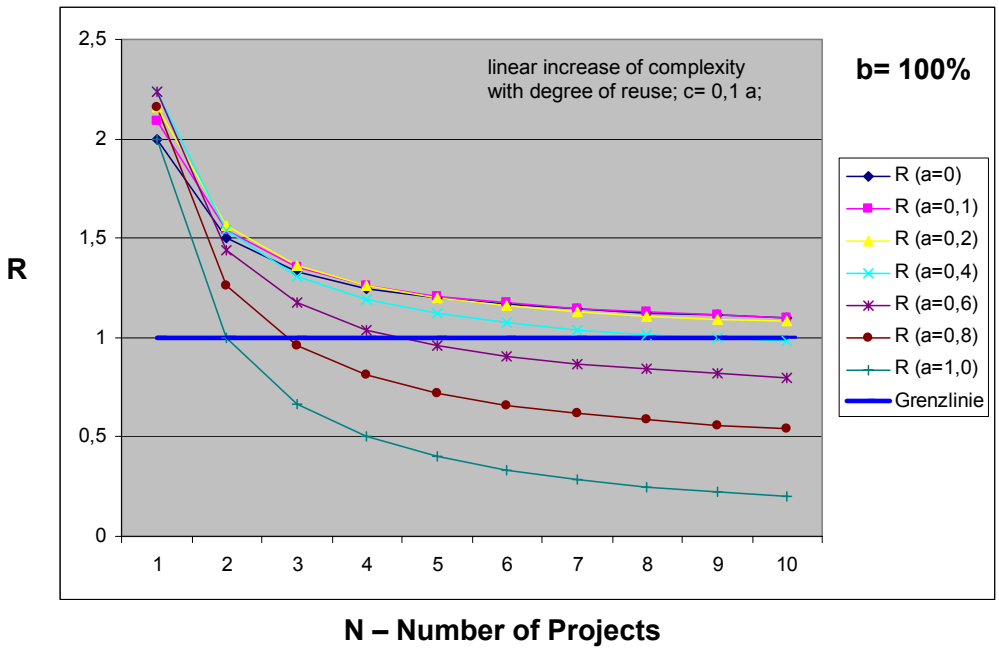

Figure 5-2: Normalized Software Development Costs per project

If we change nothing (Black-Box use case!) there is a minimum effort of roughly 5% necessary (since we must search the appropriate SWC and their configuration).
In figure 5-2 it is assumed that we change a minimum of 70% of all white box reused SWC.
It is additionally assumed that all black box reusable SWC have already been developed and are stored in an archive (this is the reason why R equals roughly 2 if we have just one project to develop).
The main results of figure 5-2 are as follows:

- A minimum of 30-40% of black-box reuse is necessary in order to reduce the SWDC per project!
- If several products (> 10) have to be developed, there is a reduction of 50% of SWDC per product if we achieve a degree of 80% of black box reuse.

In a nutshell: the introduction of a software platform or a software product line must be carefully investigated before introduction. If the scoping process shows that a degree of reuse of just 30-40% is possible or the number of expected projects (products) in small (<10), a product line or platform should not be introduced!

5.2 Target Maturity Level

The most important aspect is the definition of the appropriate **target maturity level** for each organization.

Figure 5-3 shows how the target maturity level will be defined for an organization.

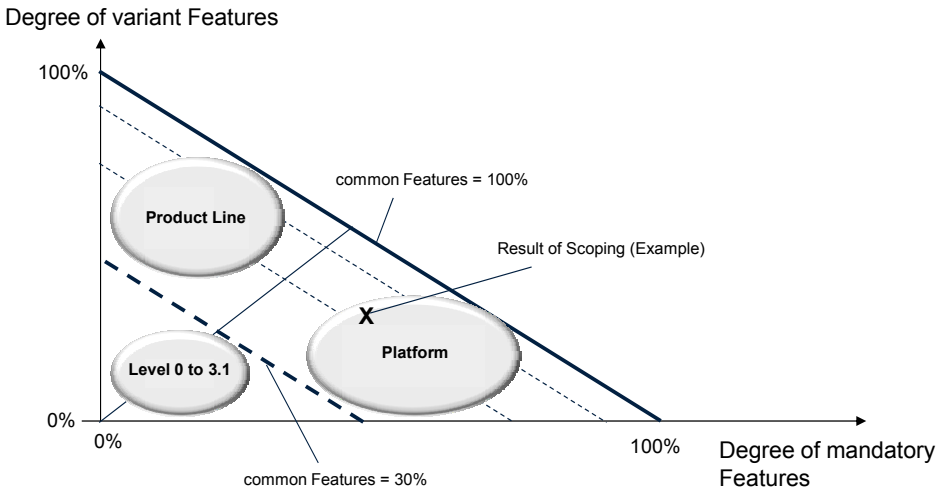

Figure 5-3: Target Maturity Level

Figure 5-3 shows a general relationship which can be applied to all automotive application domains and organizations. If the common features and therefore the expected degree of reuse is less than 30%, no effort should be spent to build up a platform or product line since the

reduction in development costs in very low (see previous chapter). Dependent on the split of variant and mandatory features (this is a result of the scoping) a platform or a product line should be chosen.
Figure 5-3 includes code-variants (from software requirements analysis to software test).

5.3 Software Development Costs

The following gives an overview on total development costs on maturity levels higher than "platform". Assumption: Each ECU software product will be developed within a project.

SWDC per year / (No. of Projects * Memory Size) = **1.000 Euro** per year / (Project*Kbyte)

The following assumptions have been used:

- SWDC includes all costs, e.g. salary of employees, tools, infrastructure
- the SWDC number is applicable for ECU software projects in the application domains Powertrain and chassis systems
- Project: activity in order to support a certain vehicle make or model or variant. The definition does not include the activity in order to build up a new ECU software platform or product line.
- Project duration is roughly 3 years

Example: ECU Memory Size = 4 MByte; SWDC per year = 4 Mio. Euro per Year / Project

This number shall be treated as a **lower limit**. It can only be achieved by organizations which have to support a high number of projects (>20) and which work on a high maturity level in mature application domain.

7 References

[1]	J. Schäuffele, T. Zurawka:	Automotive Software Engineering, Vieweg 2003, SAE 2005, PHEI 2007, Star Japan 2008.
[2]	Balzert, Helmut	Lehrbuch der Software-Technik, Spektrum Akademischer Verlag 2001
[3]	Reussner, Hasselbring:	Handbuch der Software-Architektur, dpunkt Verlag 2006
[4]	Czarnecki, K; Eisenecker, U.W.:	Generative Programming – Methods, Tools & Applications; Addison Wesley, 2000
[5]	Greenfield, Short:	Software Factories, 2004; Wiley Publishing Inc.
[6]	Seibert:	Aufwandsschätzung von Software-Projekten, TAE Esslingen, 2003