CHALMERS

Analysis of software and hardware configuration management for pre-production vehicles
– A case study at Volvo Car Corporation

Master of Science Thesis in Software Engineering and Technology

VOLHA BORDYK

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, January 2012

Analysis of software and hardware configuration management for pre-production vehicles – A case study at Volvo Car Corporation

VOLHA BORDYK

Examiner: JOACHIM VON HACHT

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden January 2012

Abstract

Modern vehicles are no longer just mechanical; most of the car's functionality is controlled by software, embedded into electronic control units (ECUs). The functionality of a vehicle is driven by customer expectations and leads to product variety since customers want their own personalized and unique car. ECU configurations consisting of different software and hardware play a vital role in providing the variability. In the automotive industry there is a tendency for shorter development cycles and several parallel projects. This combined makes keeping track of software and hardware configurations challenging, especially in pre-production vehicles where changes are introduced very often. There is also a risk that testing is performed on the wrong vehicle configuration. This can potentially lead to increasing costs and time losses. In this case study existing systems, tools and process work flows have been investigated and analysed at Volvo Car Corporation. An existing software and hardware configuration management system, used in the after market, was evaluated to what extent it could be reused. Based on the analysis, a software and hardware configuration prototype tool has been designed and implemented. The results have shown that it is possible to minimize the time necessary for preparing test setups (vehicle configurations), however limitations were found in both current processes and systems.

Acknowledgements

I would like to thank Volvo Car Corporation for the opportunity to perform my master thesis there. I would also like to thank David Sabeti Rad who have been my supervisor and all the people who have helped and supported me.

Furthermore I would like to thank the people from the Testing and Verification group and its group manager Magnus Rundgren for the support and optimism that they have given to me. I would also like to thank Gunilla Karlsson and Jonas Eriksson who came up with the idea for this thesis.

Many thanks to my supervisor at Chalmers, Joachim von Hacht, for his valuable advice and support during the work.

Thank you all!

Abbreviations and terms

Deviation Part Number - a temporary part number, used during the development phase

ECU - Electronic Control Unit

HW - HardWare

KDP - KonstruktionsData Personvagnar, product management system

Part Number - alphanumeric designation assigned to a component or product, used to identify it.

PIE - Product Information Exchange system

PP - Pilot Production

SDA - Software Download Application

SOP - Start Of Production

SSS - Software System Structure

Structure Week - production start date

SW - SoftWare

SWBL - Secondary Boot Loader

SWCE - SoftWare Configuration Element

SWLM - SoftWare Load Module

SWP - SoftWare Parameter

VBF-file - Versatile Binary Format file

VCATS - Vehicle Configuration And Test System

VCC - Volvo Car Corporation

VIDA - Volvo Information and Diagnostic Aftermarket system

VIN - Vehicle Identification Number

VP - Verification Prototype

WSDL - Web Service Definition Language

Table of contents

1. Introduction

1.1 Background

Because of the competitiveness in the automotive industry, companies must ensure that their products are of high quality. Today's modern vehicles are no longer just mechanical; most of the car's functionality is controlled by software. It is software embedded into electronic control units (ECU), also known as nodes, that provides the necessary features. Since the functionality of a car is driven by customer expectations the number of features is growing at a constant pace, and as a result, the number of ECUs is also increasing rapidly. Today a vehicle contains around 70 control units, but this number is expected to continue growing (see figure 1.1) [16]. In addition the complexity of the software embedded in an ECU is increasing [3]. Software testing is thus becoming one of the most important and crucial factors in the car development life cycle. According to surveys, companies dedicate around 50% of the software development time and cost for testing [2, 3].

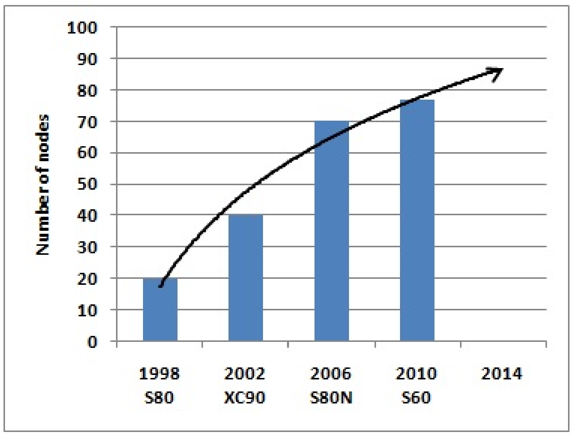


Figure 1.1: The trend of increasing the number of nodes in a vehicle

The main purpose of software testing is to improve quality. Software quality is hard to define and can include to what extent software corresponds to specified requirements and satisfies customer needs. Customer satisfaction is often considered as the main quality factor, but it is rather difficult to measure, since it usually comprises company branding as well as quality of provided services [1]. One quantifiable measurement that we can limit ourselves to is the number of defects.

Defects cause incorrect results or unexpected behaviour of the software. It is also very important when defects are discovered; before or after release of the software product. Post-release defects are usually detected by the end users and affect customer satisfaction to a large extent. They are also more expensive for the companies, since they should be fixed via updates or guarantee releases [14].

Volvo Car Corporation (VCC) performs most of its product development in the form of a project(see figure 1.2). Project starts approximately 40 months before the first car leaves the plant (milestone G1). The project includes several phases: Product Development and Manufacturing Engineering (ME). In the Product Development phase different technical solutions are created. This includes the overall design together with the internal communication networks, requirements specification, etc. At the end of the phase project strategy is conformed (G2). After this stage, VCC involves suppliers, who performs system and component development. Part of these components are ECUs. This phase is usually performed in several iterations. When acceptance and integration testing is finished, Manufacturing Engineering begins (G4). The goal of this phase is to tune and verify that the plant will be able to manufacture cars without delays; for example that all software can be downloaded into a car within 6 minutes. The final milestone is Start of Production (SOP) when everything is ready for manufacturing.

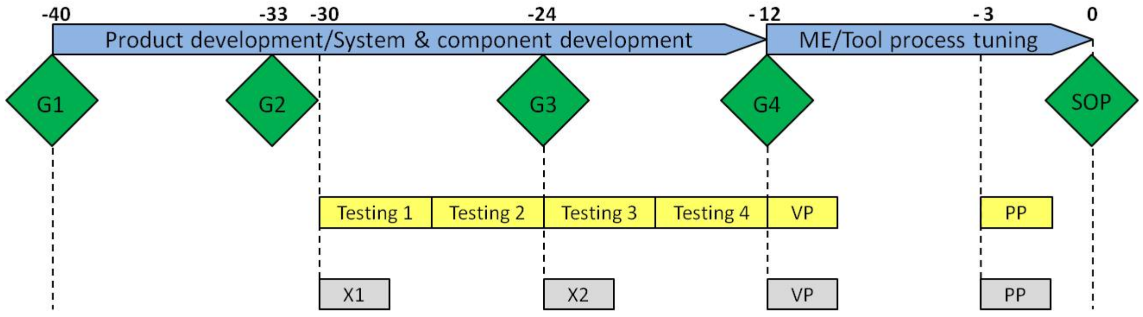


Figure 1.2: Project development plan

Several vehicle models are built on top of the same technical platform. The platform represents the development context in which vehicles share the same functions, components and suppliers. This reduces the cost of design and manufacturing.

Testing is usually performed on two different kinds of test objects, a boxcar or a regular car. Boxcars are created in the early stages of development, mainly for electrical verification. A boxcar contains just the ECUs, sensors and wires without any body parts, engine or transmission. The first car prototype (X1) is built to demonstrate system specific functionality, the body part for this prototype may not have been specified. The first full prototype X2 is build around 24 months before the start of production. The verification prototype (VP) and pilot production (PP) prototype are build at the plant to verify that the process is ready for production and after sales support.

As mentioned before, VCC outsources software development to suppliers and only performs integration and testing of software components. To be sure that the functionality corresponds to the requirement specification and legal requirements, VCC performs extensive testing on several different levels: unit, integration, functional and system testing. The trend is to start testing in the early phases of prototype cars development (Testing 1-4, see figure 1.2). This puts new demands on testing efficiency since test configurations are more volatile. Müller et al.[22] mentions research by Wehlit indicating that up to 100 changes per development day may be introduced.

In addition to actual testing, test engineers also spend a lot of time keeping track of software and hardware configurations when updating their test cars. Depending on which market a car model will we sold to, a standard car configuration can differ a lot, for example, right or left hand drive, engine type, language displayed on the screen, etc. A customer can also add extra features when buying a car; cruise control, rear park assistance, a city safety system. All these features define what hardware and software the car should contain. In addition, the same car models, produced during different period of time, can have different hardware. Software is developed for each generation of hardware and may not be compatible with the previous one. If the wrong software is downloaded into a car, it can lead to malfunction of the recipient ECUs. Because each variant needs to function properly, including all combinations, "testing and release constitute complex tasks within the overall development process"[22].

Currently test engineers have to manually configure software packages before downloading them to a test object. Before the test engineer can download a package, she also needs to manually check multiple databases for what configurations that are required for the intended test object depending on what hardware is connected to it. The information about software and hardware configurations are also manually entered into the databases by ECU managers. Both processes are time consuming and errors in these processes can potentially cause a lot of problems later, e.g. if further testing is performed on an incorrect configuration.

1.2 Problem

Today it is complicated and time consuming to update test cars with correct software. The current situation is not satisfactory and VCC tries to simplify the routines used for updating test cars.

1.3 Purpose

The purpose of the thesis is to automate configuration and distribution of the correct software packages to a test object. By doing so we will minimize the risk of verifying the wrong software and unnecessary delays during test setup. More time can thus be spent on the testing itself and consequently the number of defects detected during development will increase. As a result fewer post-release defects will reach the customer.

1.4 Method

To solve the problem existing systems, tools and process work flows will be analysed. Since some systems do not have updated technical documentation, the project will be based on close interaction with people who are responsible for their development and maintenance.

Based on the analysis, a prototype of the software and hardware configuration tool will be designed and implemented. The prototype will be of throw away nature and used only as a proof of concept to investigate what limitations exist in the systems and what changes are needed to create an efficient tool. Further improvements will be proposed as well.

Throughout the project we will use an iterative software development process instead of more traditional like the waterfall model.

2. Technical background

2.1 Vehicle electrical system

2.1.1 The electronic control unit

An electronic control unit (ECU) is an embedded computer system that is used to control and regulate various functions in a vehicle. The ECU consists of both hardware and software and is connected to a network (see section 2.1.2), and other components such as sensors and actuators. Sensors are responsible for converting a physical input (e.g. speed, temperature) to electrical signals which can be processed by ECUs. Whereas actuators are devices which convert an electronic signal to motion.

We can consider ECU hardware as a micro-controller which contains two kinds of memory: flash and RAM memory (see figure 2.1). The flash memory is non-volatile, i.e. it does not lose data when the power is switched off. It contains the primary boot loader that can not be overwritten, but other parts of this memory can be erased and reprogrammed. It is here the program code which defines the behaviour of the ECU is stored [5]. The other kind of memory, random access memory (RAM), is a temporary storage that provides better performance when programming the ECU [6].

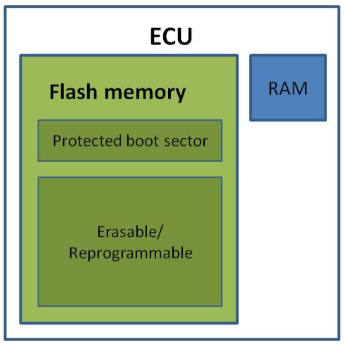
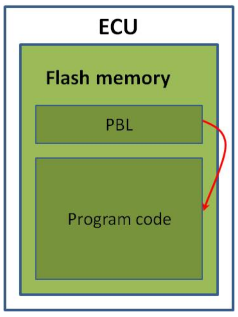


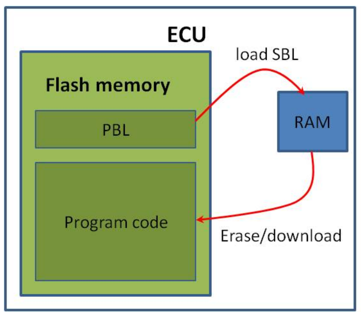
Figure 2.1: ECU hardware memory map

When starting, the ECU first executes the boot loader. The boot loader is divided into two separate parts: a primary boot loader (PBL) and a secondary boot loader (SBL). When the ECU is started in a default mode, PBL loads the application software stored in the flash memory (see figure 2.2a). Since PBL has several requirements placed on it, such as small memory size (16k) and that it can not be modified after the unit is produced, it is not suitable to be used for programming or updating data in the flash memory. The SBL is instead used for these purposes. The SBL is downloaded by the primary loader into RAM and is then activated.

After that it becomes responsible for the management of the flash memory (erasing or reprogramming) and the software download process [6], as shown on the figure 2.2b.



(a) default mode



(b) programming mode

Figure 2.2: ECU's execution modes

2.1.2 Vehicle networks

ECUs communicate with each other via networks (busses). In a vehicle there are several types of networks; CAN, MOST and LIN (see figure 2.3).

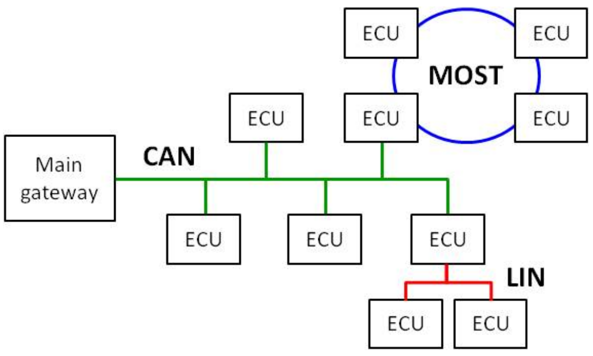


Figure 2.3: Vehicle's network scheme

Controller Area Network (CAN) is a communication network introduced by Bosch GmbH specificly for the automotive industry in 1985. CAN is a message-based broadcast protocol [4].

Since CAN has maximum bandwidth 1 Mbit/s, it is not enough to transfer video and audio data, Media Oriented Systems Transport (MOST) is used for these purposes [7]. MOST is a fiber-optic ring network, designed for efficient transmission of large amount of data in automotive systems [8, 9]. Nodes in a MOST network can not be accessed directly, just via special CAN ECUs that acts as gateways.

To reduce CAN load,Local Interconnect Network (LIN) is used to create a subnetwork on CAN for communication of low-performance ECUs. The access to LIN is done via the CAN master node that controls slave nodes in this network [10].

2.1.3 Test architecture

Communication between a tester and a vehicle can be considered as a client-server model. The information exchange between a tester (client) and an ECU (server) is performed through massages, called diagnostic services [11]. A tester can obtain different information and change an ECU state, by sending request messages. When an ECU receives a request message from a tester, it should send a response. The ECU sends a positive response if the received message was processed successfully, otherwise, a negative response is sent.

The ECU can be in different modes (sessions) that define available services; to read out information, the ECU should be in default session, whereas ECU programming is done in programming session.

2.1.4 The software download process

To download data, the ECU should be put in programming mode. First the SBL is downloaded into RAM by means of PBL (see section 2.1.1). After SBL is activated, a tester sends a service to clear the flash memory. The next step is the download of data files. Finally the ECU is reset to erase RAM and put back in default mode.

2.2 Product structure

To be competitive in the market, companies aim to develop products that on one side "meets or even exceeds customers expectations", and on the other side is easy to produce and maintain [12]. In order to satisfy today's customer needs, companies try to deliver a unique and custom made product for each customer. As a result the variety of products increases, which in turn leads to the growth of product complexity and design costs. VCC has solved this problem by using a product structure in their production.

The product structure shows components, systems, products and relationships between them that are included in a final product, in our case, a vehicle. An example of a component can be a front door which is assembled at the plant. Whereas a GPS navigator can be an example of a product purchased from another vendor.

The structure is represented as a tree, where branches are systems or subsystems (see figure 2.4). Each level in the tree represents an abstraction level; as the size of the tree grows so does the complexity of the product.

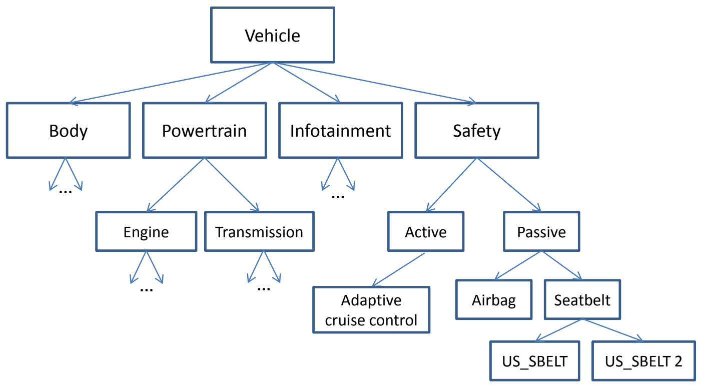


Figure 2.4: Example of a product structure

Branches are used to divide the product into standardized modules that can be reused and combined in different ways to achieve variety. In addition, a part of existing components can be used during development of new products that will reduce the cost and time-to-market [13].

2.2.1 Software system structure

Besides a product structure VCC uses a separate Software System Structure (SSS), which is a variant of the product structure for keeping track of ECU hardware and software components, also called parts. In order to introduce the reader to the context of the thesis work, this section describes the SSS in more detail.

Each year VCC introduces new changes in hardware and software because of customer demands to existing car models. Each change brings the car model to a new version, as time passed the number of ECUs increased and it became difficult to keep track of the software configurations (see figure 2.5).

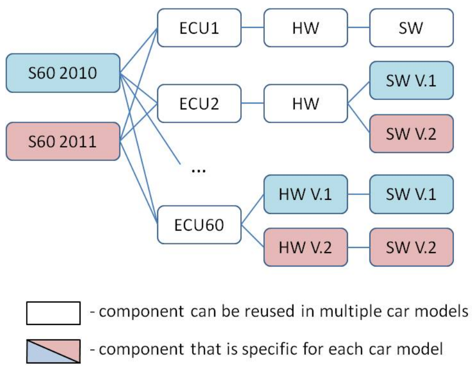

ECU1
HW
SW
S60 2010
SW V.1
ECU2
HW
S60 2011
SW V.2
...
HW V.1
SW V.1
ECU60
HW V.2
SW V.2
- component can be reused in multiple car models
- component that is specific for each car model

Figure 2.5: Example of component variability of the S60-car model, produced during different years

The SSS was developed for the customer service to support correct software upgrades in service shops. The structure connects hardware components with the appropriate software components (files) in an orderly manner. It consists of the following segments: a generic structure and a part structure.

The generic structure connects hardware and software components using part types. Since data downloaded to an ECU may contain different kinds of information, e.g. configuration or communication settings, there are several types of software components. There are also different hardware part types depending on if the hardware is an assembly part or a spare part. See table 2.1 for description of both hardware and software part types.

Part type	*Description*
HWND	this type of hardware is used both for assembly and spare parts
HWN2	this type of hardware is used if there is a variant of the spare part that contains other software files than the original one
SWBL	Software Bootstrap Loader, also called Secondary Bootloder is downloaded into a node in order to enable download of other files.
SWLM	Software Load Module file includes executable application software. When the node is running in the default mode, this software is executed, realising specified functions
SWCE	Software Configuration Element contains information about the parameters used for communication between nodes in the relevant for an ECU network (e.g. CAN or MOST)
SWP	Software Parameter file contains information about parameters used locally by SWLM file in the ECU, e.g language settings

Table 2.1: The description of part types

The generic structure is created in the early stages of development. First, it is defined which ECUs will be included in the specific platform. Then for each ECU it is specified, which hardware and software part types it should have, as shown in figure 2.6.

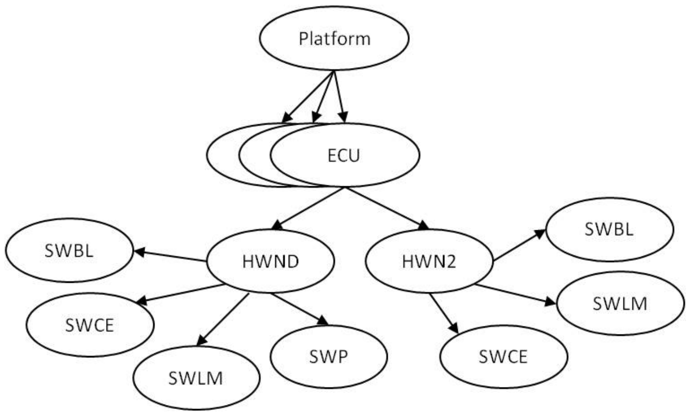


Figure 2.6: Example of the generic structure

The part structure includes part numbers (alphanumerical designations), qualifiers and time intervals during which part number is valid. This information is required when assembling vehicles at the plant. Since vehicles that belong to the same platform can vary, e.g a vehicle can have either a petrol or a diesel engine, left or right side steering wheel, there can be several part numbers for the same HW or SW part type. A qualifier is an attribute that defines which part number is valid for a particular vehicle configuration. When new part numbers are introduced, they are connected to the generic structure and we get the complete SSS, as shown in table 2.2).

For example, the driver door module shares the same HW part number (111111) independently if the vehicle is a left- or right-hand driven, as indicated by the HW qualifier (LHD,RHD). The same is true for the SWBL and SWCE software part types (222222 and 333333). Whereas for the SWLM software part type the qualifier specifies two different software part numbers, 444444 and 555555 for LHD and RHD respectively.

ECU	*Year*	*Level*	*HW Part No*	*HW Part Type*	*HW Qualifier*	*SW Part No*	*SW Part Type*	*SW Qualifier*
ECU1	2011	prod	111111	HWND	LHD,RHD	222222	SWBL	LHD,RHD
ECU1	2011	prod	111111	HWND	LHD,RHD	333333	SWCE	LHD,RHD
ECU1	2011	prod	111111	HWND	LHD,RHD	444444	SWLM	LHD
ECU1	2011	prod	111111	HWND	LHD,RHD	555555	SWLM	RHD

Table 2.2: An example of the complete SSS for the driver door module, where RDH and LHD are right- and left-hand drive respectively

The SSS has two maturity levels; verify and production. The verify level is used for verification purposes in the later stages of development. The other level is used when the structure is actually released to production.

3. Analysis of existing processes, systems and tools

In this chapter we analysed the existing processes, systems, tools and elicited the requirements for the future tool. The analysis resulted in the creation of a basic idea for a future design.

3.1 Current process

Today the information regarding software configurations, test orders and actual software files are stored in several places; in Lotus Notes, on network shares or the Software Archive (see figure 3.1). Lotus Notes is a document management system developed by IBM to enable information spread within the company [21]. Both on the network shares and Lotus Notes the information regarding software configurations and test orders is saved as Word or Excel documents. As mentioned in section 1.1, car configurations can differ, thus each test vehicle has a specific configuration file, called CarConfig, which is also stored there.

ECU managers are responsible for entering and updating information for different projects both in Lotus Notes and on network shares. After the software have been approved at the Change Request Board, files are manually moved to the Software Archive, a storage for software files used in production and by the aftermarket. The manual handling process results in human errors, e.g. different software files can be assigned the same part number. In addition, the distributed storing causes information inconsistency, since the information can be not updated on time or in the right places.

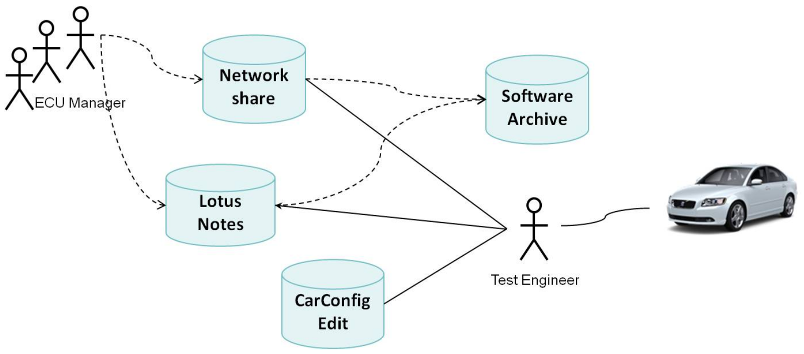


Figure 3.1: Current process overview

Before each test, a test engineer gets a test order. The test engineer reads out data from a test vehicle to identify if a test vehicle has the right configuration. If needed a CarConfig file is edited by means of the CarConfig Edit application.

The Test engineer also checks if the correct version of hardware is mounted in the test vehicle and retrieves software files specified in a test order. Both Lotus Notes and the network shares have a folder structure and does not provide a good search functionality. After CarConfig and software files have been downloaded to the test vehicle, the test engineer can start to perform tests. The described actions are time consuming and causes time delays and thus results in extra costs during the development process.

3.2 Requirements elicitation

As it was mentioned before, the project was based on close interaction with test engineers and people responsible for the development and maintenance of existing systems and tools. During meetings the requirements of the SW and HW configuration tool were elicited. Further the requirements were analysed and Software Requirement Specification (SRS) was created (see Appendix A).

3.3 Information systems

3.3.1 Product management system

VCC uses a product management system which is called Konstruktionsdata Personvagnar (KDP). KDP was developed in the 70's when a vehicle was more mechanical than electrical. The system runs on an IBM mainframe and includes DB2 databases and a Query Management Facility (QMF) interface for SQL queries [18]. The query results can also be exported as text or Microsoft Excel files.

The system is intended to manage and keep track of products during different stages of their lifecycle; design, development and maintenance. KDP is also a provider of information for other systems in factories, to suppliers, retailers and the aftermarket. The system is divided in a number of modules, but we analysed only two of them; Parts and SSS.

The Parts module contains data about parts that will be assembled in the factory (the part structure); what raw materials and components are needed and their quantity to mount a car. To minimize assembly time some parts are arranged in so called delivery units. For example, software can already be downloaded into an ECU before it will be mounted into a car. Thus software can be a stand alone unit or a part of a delivery unit in the Part structure.

When the number of ECUs increased, part structure was not appropriate to keep track of SW and HW configurations for software updates in the aftermarket. For this reason, the SSS module was introduced in 2003. The SSS structure is created based on the information from the part structure and generic structure, as described in section 2.2.1. A complete SSS structure is available around 3 weeks before start of production and is transferred to the PIE system (see section 3.3.2) on a weekly basis via internally developed communication tools. The complete SSS contains just so called “start of production” (SOP) part numbers, permanent part numbers that cannot be changed after the production starts. In the development phase deviation (temporary) part numbers are frequently used.

3.3.2 Product Information Exchange system

The Product Information Exchange (PIE) system is an aftermarket IT system for SW storage and vehicle's software update handling. It is a web-based application built on J2EE and an Oracle database. It provides communication with other systems via web services using, as described in the Web Services Description Language (WSDL).

The PIE system has a module structure and consists of the following subsystems: Software Archive (SWA), Vehicle database (VEH), Software Product (SWP) and coordination module (SAFE) (see figure 3.2).

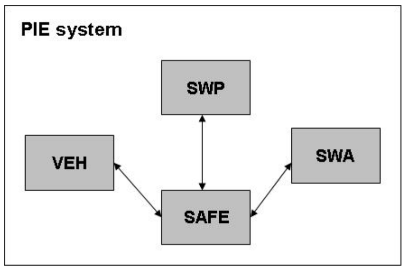


Figure 3.2: PIE system structure

The Vehicle Database contains information about HW and SW configuration for each particular car. The information about a car configuration is transferred from the factory when a car is produced. Moreover, the history of configuration changes is stored in this database each time when new software is downloaded or an ECU is replaced at the service station. The SSS is stored in the SWP subsystem (see chapter 2.2.1). The Software Archive is a storage area for all software files.

At a service station the car is connected to the PIE system via the VIDA application (see figure 3.3). Each car has a unique identifier, known as the vehicle identification number (VIN). When the vehicle is connected to VIDA, VIN is read from the vehicle and sent to PIE to check if the vehicle is registered in the VEH database (1). To get the software files, the technician should also provide a so called Software Product (SP) number that represents what kind of modifications will be performed, e.g. a software upgrade or ECU replacement (2). According to the provided SP number a recipe is retrieved from the SWP module. The recipe specifies which software file types should be loaded for this particular modification. The correct software file numbers are retrieved by checking the SSS in SWP (3). Then the correct software files are found in the SWA module and downloaded to the car using VIDA. If the modification is successful, changes are registered in the VEH database.

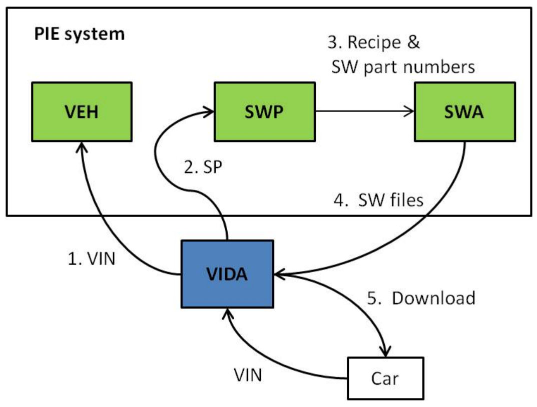


Figure 3.3: Software update process in the PIE system

3.4 Software download tools

At VCC today, the download of software files is managed using the following tools; VCATS, VIDA, SDA. By analysing different tools and their pros and cons, a suitable software download tool was chosen for the prototype.

3.4.1 VIDA

Volvo Information and Diagnostic Aftermarket (VIDA) is an application which is used to support the repairment and service of Volvo cars at service stations. The application includes the following functionality; information about spare parts, descriptions of repair procedures, vehicle diagnostics and software updates. VIDA is a client application. Some information like a spare parts catalogue and repairment descriptions are stored locally during installation and works without internet connection. But connection is required in order to perform software updates. In that case VIDA connects to PIE via VPN over the internet to get software files (see section 3.3.2).

To assist the diagnostic technician and reduce the service time, VIDA automatically sends necessary diagnostic services to the vehicle and presents the information regarding faults in a simple way on the screen. The software update process is also automated; a technician should only choose an ECU and an action which should be performed, e.g ECU upgrade or replacement. Then VIDA itself will send all necessary information to the PIE system and perform a complete software download. When the download is completed the new status of the car is reported back to PIE.

3.4.2 VCATS

Vehicle Configuration And Test System (VCATS) is used specifically just for software download and diagnostic testing at the production plant. It was developed to assist plants in building cars. The system is also used by test engineers at a later stage of the development to verify that the download process is ready for starting of production.

The system has a client/server architecture. The server is connected to test stations (clients) and the production plant data system. The plant data system process a customer order, generating a unique control data string for every car, which is transferred to the VCATS server. Based on the received data, the server retrieves download scripts and software files. The server also generates a test sequence. Then this information is sent to a test station, where the operator downloads software and performs tests. Tests are performed to verify that download process went well and all ECUs are working.

VCATS is a fully automated tool; download scripts are created in advance and stored on the server. The user just need to read out the car id (VIN) and press a button to start the download process; the system will perform the whole job automatically. In addition, the tool has the capability of downloading files much faster, because of parallel downloads, than the development (SDA) and the aftermarket (VIDA) tools. However, the tool is not used for software updates of test objects during the development, since a lot of time is required to create download scripts.

3.4.3 SDA

The Software Download Application (SDA) is used by test engineers for software downloads during the development phase. SDA is a desktop application built in Visual Basic. The application is standalone and has no connection to any other systems.

The application has two download modes. An automatic mode where the process is defined in a configuration script, following the description in section 2.1.3, and the other where the user manually performs the download steps. In the latter case, the test engineer can control and follow the download process.

The application also provides the functionality to read out data from a vehicle, using predefined read services. Since it is only possible to send one service at a time, this functionality is rarely used by test engineers. For example, a test engineer might need to read out information about all hardware part numbers in the vehicle, then she needs to send the same service to each ECU.

The application provides an external interface (API) using Microsoft component object model (COM) [20]. The API exposes methods that represents different diagnostic services.

3.4.4 Analysis

In previous sections we looked upon different software download applications currently utilised at VCC. In this section we analysed applications pros and cons concerning the requirements as specified in appendix A, while choosing the appropriate tool to be used for software download.

As mentioned before VCATS requires the preparation of download scripts. Since there are several software releases during the development phase, it will require a lot of human effort. It was not appropriate in our case, since one of our goals was to minimize the maintenance workload. Thus we decided not to use the VCATS application for software download.

In most cases test engineers just want to update software in a car and then perform tests. But in some cases they need to control the download process to identify a problem or to verify that the the process is going according to the specification. The main advantage of SDA over the VIDA application is that it provides both an automatic and manual software download functionality, whereas the VIDA applications is fully automated. In addition, VIDA requires a constant online connection with the PIE system during the software update, since it sends confirmations back to PIE. It is a drawback for off-line testing, when test engineers are e.g. on expeditions. In that case, files should be obtained in advance and downloaded when needed.

Due to the reasons mentioned above, we decided to use SDA as a software download application in our prototype.

3.5 System Architecture

Based on the specified requirements, we defined two areas for the implementation work; the first one was to generate the SSS including deviation part numbers and import it to the PIE system, the other one was to integrate the PIE system and SDA tool to get the application that will be used directly by test engineers (see figure 3.4).

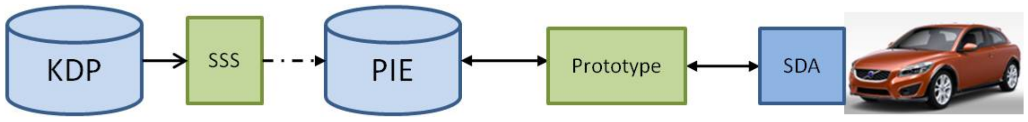


Figure 3.4: Architecture overview

We also defined the flow of data between the PIE and SDA (see figure 3.5). The information about hardware part numbers and car identification number (VIN) will be read out from a test object, using diagnostic services via SDA tool. This information will be sent to the PIE system to receive software files and access codes. Later, files will be downloaded to a test object via SDA.

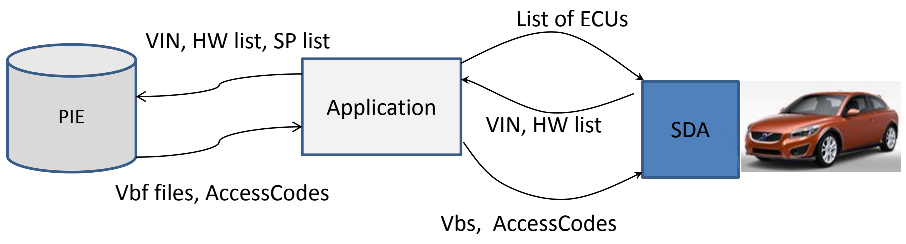


Figure 3.5: The data flow

3.5.1 Limitations

Since the main purpose of the prototype is the proof of concept we decided to put some limitations:

- The prototype will support just one platform that includes the following car models; S60, V60 and XC60.
- The prototype should generate a valid SSS starting from VP series (see section 1.1).
- The SSS will not include information about accessories parts.
- The tool will work just inside the VCC Intranet.
- No changes will be introduced to existing systems.

4. Prototype implementation

Java technology was chosen as a platform for the prototype implementation. The choice was made based on that Java is a platform-independent and open-source technology. Furthermore, there are a lot of open-source projects that promote the integration of Java with other technologies [17]. As a development tool we used the Eclipse IDE. During the implementation we followed the principles of the object-oriented programming (OOP).

4.1 General design

We started from the design of the application that will integrate PIE and SDA. We defined two package roots; ecuswhandling and webservice (see figure 4.1). The ecuswhandling package groups the packages and classes for the client. It follows the Model View Controller pattern. The webservice package contains the stub for the webservice interaction.

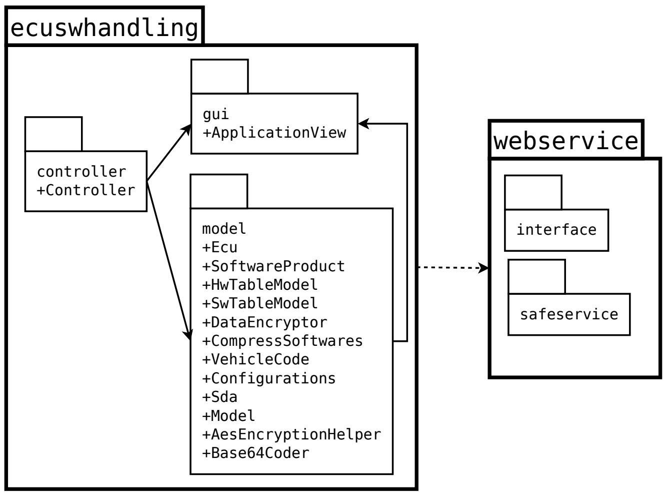


Figure 4.1: Application package architecture diagram

4.2 Iteration 1. The generation of the software system structure

The initial idea was to generate the SSS inside the KDP system using query management facility (QMF) (see section 3.3.1). This however turned out to be both risky and costly. Each QMF-query costs money to execute on the mainframe and there was no budget allocated to our project. KDP is also a truly critical system that Volvo could not let us access. Thus we decided to create a solution that will be compatible with KDP but external to it.

As mentioned before, the results of SQL queries from KDP can be stored as Microsoft Excel files. We made a decision to take one of such files with the part structure and deviation numbers that is generated on a weekly basis. We developed a desktop application that takes the file as an input, process it and generate the file with the SSS that can be easily imported into the PIE system. We used a local SQLite database to process data using SQL queries.

Since the data in the file had a tree structure which was not suitable for direct importing into the relational database, we needed first to filter it. For this we used the Java API for Microsoft documents developed within Apache POI project [19]. After the data was imported, SOP part numbers in the PartStructure table were updated with deviation numbers.

The generation of the SSS was performed in two stages. The first one consisted of getting valid part numbers for the hardware and software types defined in the generic structure. For this reason a database schema was created to store the cope of the generic structure (see appendix B). Two views were created, HwPartNoView and SwPartNoView, based on retrieving information from the PartStructure, NodeHw and NodeSw tables. During the second stage hardware and software were merged based on the qualifiers in the MergeSwHw view and output file with the SSS was generated.

During this iteration a simple graphical user interface (GUI) for the application was also developed using Java Swing. The main window contains fields where the user specifies for which structure week, project and series the SSS will be generated. Using the File menu the user opens the input file with the Part structure and chooses a location to store the generated file with the SSS.

4.3 Iteration 2. Interaction with the COM object

As mentioned before, the SDA tool provides an API using COM (see section 3.4.3). There are both commercial (EZ-JCom, Java2COM) and open-source projects (JAWIN, JACOB, COM4J) that enable the integration of Java and COM technologies. Since our project did not have a budget we investigated just open-source projects and chose the JAWIN implementation. During the investigation of different projects JAWIN was easy to get started with because of good documentation. It also showed good results in memory consumption and fulfilled all our requirements. The library contains the implementation of Java Native Interface (JNI) that automatically converts Java classes into native code, thus no additional compilations are needed [15]. JAWIN provides the library as a jar file that can be easily added to the Java project.

The SDA API provides methods for communication with a vehicle by means of diagnostic services. To initiate the communication, the initialisation method was called that specifies the application setup. The SDA tool uses a configuration xml file that contains information about different platforms, bus types and ECUs, specified for each platform. We decided to use this file so that we can later get the information required to call the API method that represents the read out service. Calling this method, we got VIN and HW part numbers in a hexadecimal format for the ECUs that are present in the vehicle. After parsing and converting into the decimal format, the data were stored as an object to pass via web services to the PIE system (see section 4.4).

To access nodes in the MOST network, it is necessary to open the gateway using the method that implements the remote activation of routines in the MOST node. Calling this method we did not get the stable result, so during one of the meetings it was decided to exclude MOST nodes from the prototype implementation.

To download software files into a vehicle, the download method was used. The method provides the opportunity to download both single file and a sequence of files. Since several files are needed to update an ECU we decided to always download the sequence of files. Before performing the actual download, most of the ECUs require us to provide a security access code. By doing so, we gained access to manipulate the flash memory in the ECUs.

We also used information related methods to report errors. All error messages were saved in a log file. We used the Apache log4j library to create the log file. In addition, the SDA tool has its own log file, thus checking both log files the user can faster identify a problem.

4.4 Iteration 3. Communication via web services

The PIE system provides several environments: production, acceptance and testing. Usually test engineers are working in the acceptance environment. The interaction with other systems are performed be means of web services. From the PIE development team we got the Web Services Description Language (WSDL) file that describes the information required to use the web services. Because we only consume web services, we let Eclipse generate stub classes from the WSDL file.

To get the software files, the web service requires us to provide VIN, HW part numbers and SP numbers. SP numbers were stored in the XML file that has the following schema:

```
<?xml version=”1.0” encoding=”UTF-8”? >
< SoftwareProduct >
   <Configuration Name=” ” >
       <ECU Name=” ” Address=” ” SP=” ” >
       <ECU Name=” ” Address=” ” SP=” ” >
   < /Configuration >
   <Configuration Name=” ” >
       <ECU Name=” ” Address=” ” SP=” ” >
       <ECU Name=” ” Address=” ” SP=” ” >
```

$< /Configuration >$
$< /SoftwareProduct >$

Software files are transferred over the network both zipped and encrypted. After receiving the files, they were extracted and decrypted using a library. After that the files are ready to be downloaded into the vehicle. We identified a bug in this process but for obvious reasons no further detail will be provided.

4.5 Iteration 4. Development of the graphical user interface

During this iteration the graphical user interface (GUI) of the application was developed. The functional requirements were defined and written down to the specification as described in appendix A. During the requirements elicitation stakeholders did not put specific requirements regarding how the GUI should be designed; it was up to us to decide on the application appearance. We decided to include all information in one screen since it should be easy for the user to access and view all the information about a test vehicle. One of the negative aspects of putting all information in one screen is that the user can be overwhelmed. Since there was not so much information, we did not face this problem. The users will also become experts by using it very frequently. More information about the main screen is provided in appendix C.

The menu tool bar was developed to provide functionality for saving software files and choosing connected devices. A traditional save function was implemented using the Java Swing API. The application also includes a status bar, dialogue and info windows to provide error messages or warnings to the end user.

5. Results

The following results were presented at the end of the thesis work.

One of our problems was to minimize the time necessary for preparing test setups (update test vehicles with correct software). Based on our empirical study of the current process (see section 3.1) it takes on average three minutes for a test engineer to extract and download software files for one ECU. While using our new automated prototype we can cut down the required time to less than one minute. Updating the whole CAN network would then save approximately 20 minutes.

Another problem was to lower the number of mistakes done by ECU managers during the creation of software configurations. By generating the SSS automatically, we minimized the required preparation time and potential for human generated errors.

6. Discussion

The prototype shows that the PIE system provides benefits for improving efficiency in collecting data, such as software configurations, test object configurations and software files from different sources. Since the PIE system will be used in earlier stages of the development, all software files can be placed directly in the Software archive. By unifying the storage of software files, we solved the problems related to inconsistency as described in section 3.1.

A potential problem can be that information used for building the SSS may not be inserted or updated on time in KDP. For example, we found out that deviation numbers for preloaded software are often not updated in KDP and stored locally by ECU Managers. In that case, it is not possible to fully automate the process of generating the SSS. One way to solve this problem is to introduce a step in the process were all necessary data will be entered and verified in KDP before generating the SSS. Another possible solution is that the SSS will be verified and edited after it was imported into the PIE system by ECU Managers. This has the disadvantage that SSS containing errors might propagate to other systems.

At the end of the project we did not deliver a fully functional tool. During the prototype development and testing we discovered a number of limitations in the way both SDA application and the PIE system works. All findings have been documented and discussed during meetings with stakeholders. The major issues are listed below:

- As mentioned in section 4.4, it is not possible to read out data from nodes on the MOST network.
- A test object should have at least three hardware components to be considered a vehicle in PIE. In that case, currently the PIE system can not be used for single component testing.
- A vehicle and information about its configuration shall be registered in PIE. Today the information about test vehicles produced at the plant is automatically transferred to the system. The information regarding test objects such as boxcars should be entered manually.
- Each vehicle shall have an identifier (VIN). Nowadays early prototype cars (boxcars) does not have a VIN number. Since a VIN number is not just a random generated number, it caries the information about the vehicle, this problem should be further investigated and a process for generating VIN numbers for boxcars should be introduced.

7. Recommendations for future work

Recommendations for future work include such things as continued investigation of discovered problem limitations and implementation of a fully functional tool.

In the beginning of the project it was decided that no changes will be introduced to existing systems. We think it would be beneficial to extend the SSS in the future. One of the possibilities would be to include information regarding projects and series (i.e. releases), since this information is associated with important milestones in the pre-production development phase. This would not affect the after market system because it ignores excessive data.

Introduction of more maturity levels could be another possible extension. Currently the SSS has just two maturity levels, verify and production. For example, levels when software is approved for component-, integration- and functional testing. Software for integration testing should be available only after it passed component testing. By introducing these levels and enforcing that the are done in order, by putting up process gateways, we could minimise the cases when testing is performed in the wrong order.

Another possibility would be to extend the SDA application so that it will communicate directly with the PIE system. That is, to integrate our application into SDA. By doing so we can exclude one unnecessary connection. It would enable a test engineer to perform the whole software update process automatically from SDA; read out information from a vehicle, get software files and download them to the test object. It will give more freedom for a tester that will lead to a better testing process as a whole.

Bibliography

[1] G. J. Myers, C. Sandler, *The art of testing 2nd edition.* Wiley, 2004

[2] K.Emam, *The Roi From Software Quality.* Auerbach Publications Boston, USA, 2005

[3] C. Ebert, C. Jones, *Embedded Software: Facts, Figures, and Future*, IEEE Computer Society Press, 2009. Available at: `http://www.computer.org/portal/web/csdl/doi/10.1109/MC.2009.118`, retrieved 2011-04-13.

[4] Robert Bosch GmbH, *CAN Specification*, Version 2.0 edition, 1991

[5] A. Weimerskirch, *Secure Software Flashing.* SAE International Journal of Passenger Cars - Electronic and Electrical Systems, 2009

[6] Mentor Graphics, *Vehicle system design*, 2007. Available at: `http://www.mentor.com/products/vnd/upload/VolcanoBootloader_DS.pdf`, retrieved 2011-07-29

[7] P. Smith, *Automotive gateways spearhead in-car network integration*, EE Times, 2005. Available at: `http://www.eetimes.com/design/automotive-design/4011050/Automotive-gateways-spearhead-in-car-network-integration`, retrieved 2011-06-08

[8] MSDN, *MOST network.* Available at: `http://msdn.microsoft.com/en-us/library/ms859428.aspx`, retrieved 2011-06-10

[9] MOST Cooperation, *MOST network*, 2011. Available at: `http://www.mostcooperation.com/technology/most_network/index.html`, retrieved 2011-07-30

[10] STMicroelectronics, *LIN (Local Interconnect Network) Solutions*, 2002. Availale at: `http://www.st.com/stonline/books/pdf/docs/8130.pdf`, retrieved 2011-06-10

[11] International Organization for Standardization, *Road vehicles — Unified diagnostic services (UDS)*, 2006. Available at: `http://www.saiglobal.com/PDFTemp/Previews/OSH/iso/updates2007/wk16/ISO_14229-1-2006.PDF`, retrieved 2011-06-20

[12] N. Snack, S. Chambers, R. Johnston, *Operations management 5th edition*, Prentice Hall, 2007, pp 129-131

[13] DRM Associates, *Product structure and bill of materials.* Available at: `http://www.npd-solutions.com/bom.html`, retrieved 2011-08-01

[14] C. Stringfellow, A. Andrews, C. Wohlin, H. Petersson, *Estimating the number of components with defects post-release that showed no defects in testing.* Software Testing, Verification and Reliability, 2002, pp. 167–189.

[15] Sourceforge, Jawin - a Java/Win32 interoperability project, 2005. Available at: `http://jawinproject.sourceforge.net/`, retrieved 2011-02-23

[16] E. Johansson, S. Romero Skogh, H. Svensson, *Software management within Product Development*, Chalmers University of Technology, Göteborg, 2011.

[17] Oracle, *Oracle and Java.* Available at `http://www.oracle.com/us/technologies/java/java-se-405465.html`, retrieved 2011-08-17

[18] IBM, *DB2 Query Management Facility.* Available at: `http://www-01.ibm.com/software/data/qmf/`, retrieved 2011-08-20.

[19] Apache POI project, *Apache POI - the Java API for Microsoft Documents.* Available at: `http://poi.apache.org/`, retrieved 2011-03-15

[20] Microsoft, *Component Object Model Technologies.* Availavle at: `http://www.microsoft.com/com/default.mspx`, retrieved 2011-04-01.

[21] IBM, *Lotus Notes Software.* Availavle at: `http://http://www-01.ibm.com/software/lotus/products/notes/`, retrieved 2011-10-15.

[22] D. Müller et al., *IT Support for Release Management Processes in the Automotive Industry*, Fourth International Conference on Business Process Management, 2006.

A. Software Requirements Specification

”Software and hardware configuration tool”

A.1 Introduction

A.1.1 Purpose

This document specifies the requirements for the prototype implementation of the software (SW) and hardware (HW) configuration tool. The specification is meant to be used to collect and describe requirements both for system developers and stakeholders.

A.1.2 Scope

The purpose of the tool is to automate configuration and distribution of the correct software packages to the test object. Currently, software handling process is manual and time consuming. The SW and HW configuration tool is supposed to be used by testers to quickly update test objects with the latest software.

A.1.3 Definitions, acronyms, and abbreviations

Term	Description
Deviation number	temporary part number used during development phase
ECU	Electronic Control Unit
KDP	Konstruktionsdata Personvagnar, product management system
PIE	Product Information Exchange, Aftermarket IT system
SDA	Software Download Application
SSS	Software System Structure
Structure week	production start date
VP	Verification Prototype
Test object	a car or a boxcar
User	a test engineer

A.2 Overall description

A.2.1 Product overview

The SW and HW configuration tool should allow the user to generate the software system structure (SSS) automatically based on the information received from the KDP system. The generated SSS should be able to be imported into the PIE system in an easy way. The tool should also be used directly by test engineers for software updates of a test object. To achieve it, the PIE system and SDA should be integrated, so the user will not realise their presence.

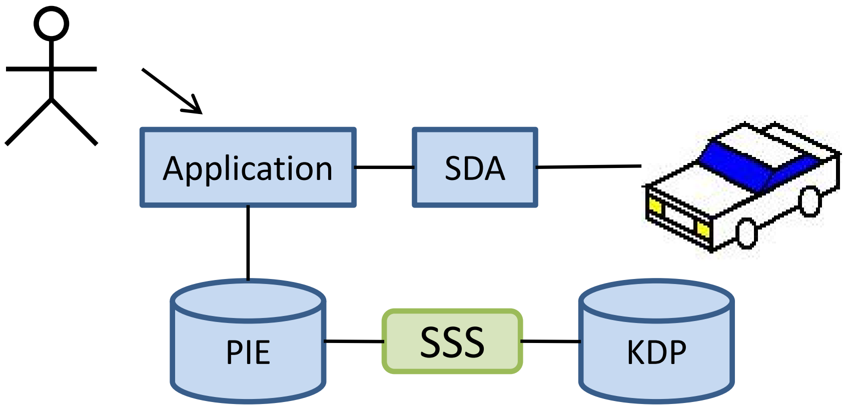


Figure A.1: The overview of the SW and HW configuration tool

A.2.2 Product functions

The SW and HW tool should handle the software update process in a test object (a car or a boxcar). The user should be able to read out data from the test object and get correct software files for current configuration. The user should be able to download the chosen files automatically or manually. The user should be able to update one or several ECUs. The tool should provide opportunity to safe software files to update the test object during expeditions when the internet connection to the server is not available.

The tool should facilitate the process for creation of the SSS. The structure should be generated automatically for a specific structure week, based on the part structure and deviation numbers. The structure should also include preloaded software. The structure should be created starting from VP series.

A.2.3 Operating environment

The SW and HW configuration tool will require that the user will be connected to the VCC Intranet to get software files. The user can read out data and download previously received files in off-line mode.

Since the SDA external interface is closely connected to the SDA application, the SDA application should be installed and work properly.

A.2.4 Constraints

Since the purpose of the prototype is the proof of concept, no security issues will be considered.

A.3 Specific requirements

A.3.1 Functional requirements

3.1.1. Read out data from the test object

Id: F.1
Description: The user should be able to read out HW numbers and VIN from the test object.
Category: Functional
Classification: Mandatory
Priority: High

3.1.2 Display hardware information

Id: F.2
Description: The tool should display name, address and hardware number for ECUs which are presented in the test object
Category: Functional
Classification: Mandatory
Priority: High

3.1.3 Selection of nodes

Id: F.3
Description: The user should be able to select one or several ECUs for which she wants to get software files
Category: Functional
Classification: Mandatory
Priority: High

3.1.4 Total reload

Id: F.4
Description: The SW and HW configuration tool should provide opportunity to update all ECUs at a time in the test object
Category: Functional
Classification: Optional
Priority: Low

3.1.5 Get software files

Id: F.5
Description: Providing VIN and hardware number(s), the user should be able to get correct software files valid for structure week the test object belongs to
Category: Functional
Classification: Mandatory
Priority: High

3.1.6 Display software files

Id: F.6
Description: Received software files numbers shall be displayed on the screen
Category Functional
Classification: Mandatory
Priority: High

3.1.7 Selection of files

Id: F.7
Description: The user should be able to select one or several software files that she wants to save locally or download into the test object
Category: Functional
Classification: Mandatory
Priority: High

3.1.8. Download files

Id: F.8
Description: The user should be able to download software files in the test object
Category: Functional
Classification: Mandatory
Priority: High

3.1.9 Get access codes

Id: F.9
Description: Providing VIN and hardware number(s), the user should be able to get access code(s) to unlock ECU(s)
Category: Functional
Classification: Mandatory
Priority: High

3.1.10 Save files

Id: F.10
Description: The user should be able to save received software files
Category: Functional
Classification: Mandatory
Priority: High

3.1.11 Communication driver selection

Id: F.11
Description: The user should be able to select a communication driver (e.g. CAN card, DICE)
Category: Functional

Classification: Mandatory
Priority: High

3.1.12 Generate SSS

Id: F.12
Description: The SSS should be generated automatically starting from VP series
Category: Functional
Classification: Mandatory
Priority: High

3.1.13 Deviation numbers

Id: F.13
Description: The SSS should include deviation numbers valid for the week when it is generated
Category: Functional
Classification: Mandatory
Priority: High

3.1.14 Preloaded software

Id: F.14
Description: The SSS should include preloaded software
Category: Functional
Classification: Mandatory
Priority: High

3.1.15 Edit SSS

Id: F.15
Description: The user should be able to edit the SSS if necessary
Category: Functional
Classification: Optional *Priority:* Medium

3.1.16 Provide feedback

Id: F.16
Description: The tool should provide the user with feedback information such as status and warnings
Category: Functional
Classification: Mandatory *Priority:* Medium

A.3.2 Non functional requirements

3.2.1 Authorization

Id: NF.1
Description: The user should be authorized

Category: NonFunctional
Classification: Mandatory *Priority:* High

3.2.2 Usability

Id: NF.2
Description: The user interface should be easy to understand
Category: NonFunctional
Classification: Mandatory *Priority:* High

B. Database scheme

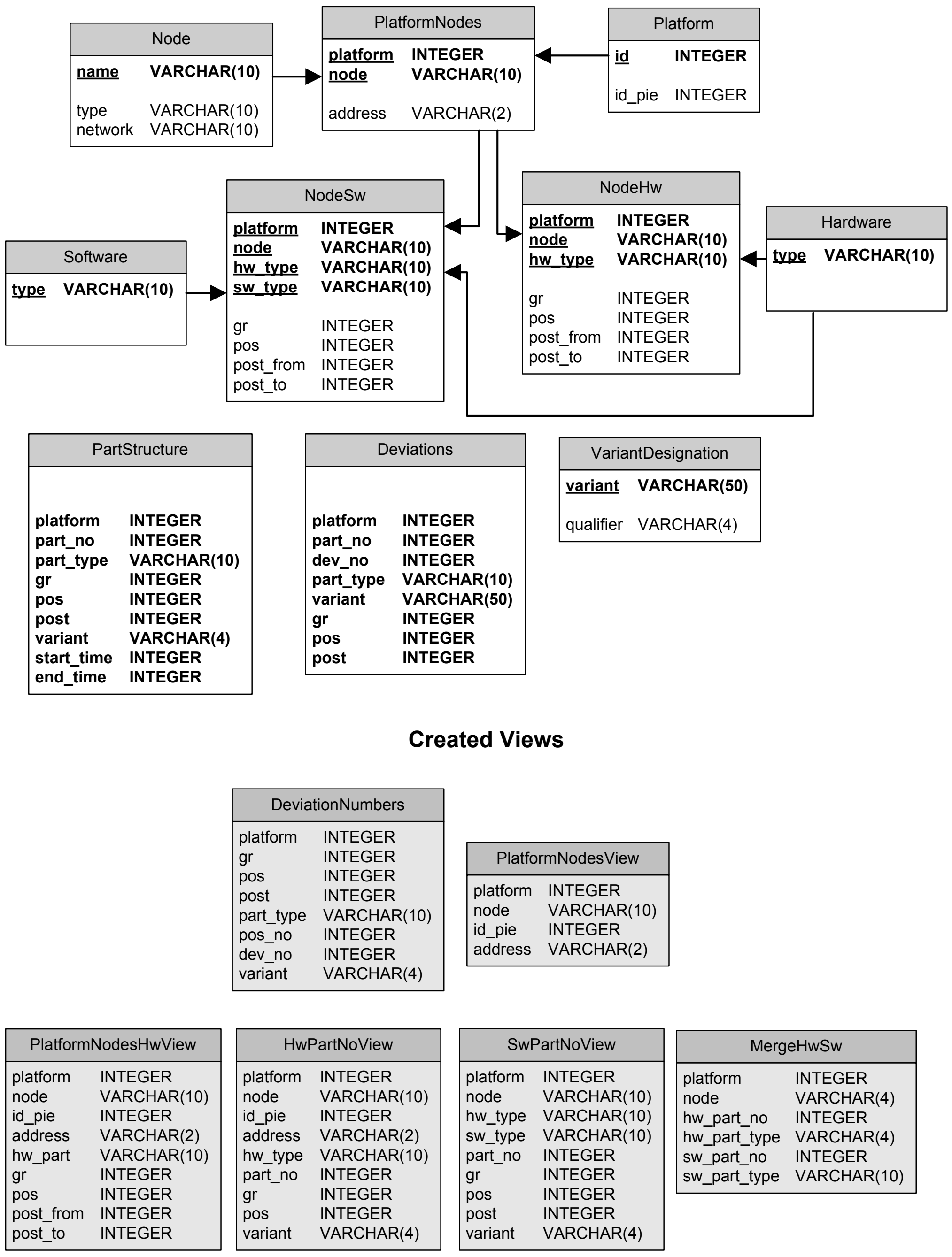
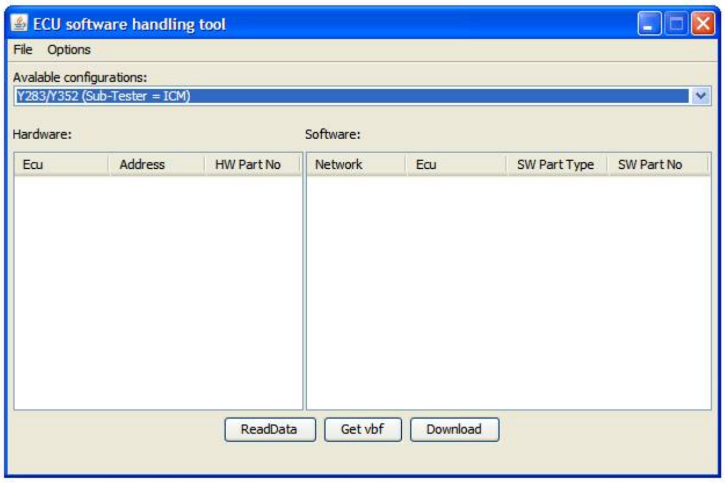


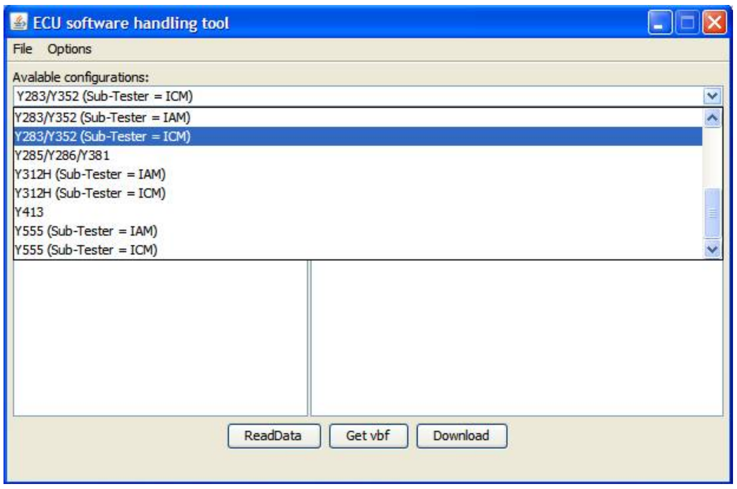
Figure B.1: Database Model Diagram

C. Prototype demo



Avalable configurations: Y283/Y352 (Sub-Tester = ICM)

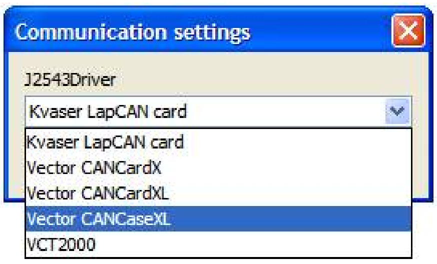
Figure C.1: The main application window that is used by test engineers to update software in the test object



Avalable configurations: Y283/Y352 (Sub-Tester = ICM)

Y283/Y352 (Sub-Tester = IAM)
Y283/Y352 (Sub-Tester = ICM)
Y285/Y286/Y381
Y312H (Sub-Tester = IAM)
Y312H (Sub-Tester = ICM)
Y413
Y555 (Sub-Tester = IAM)
Y555 (Sub-Tester = ICM)

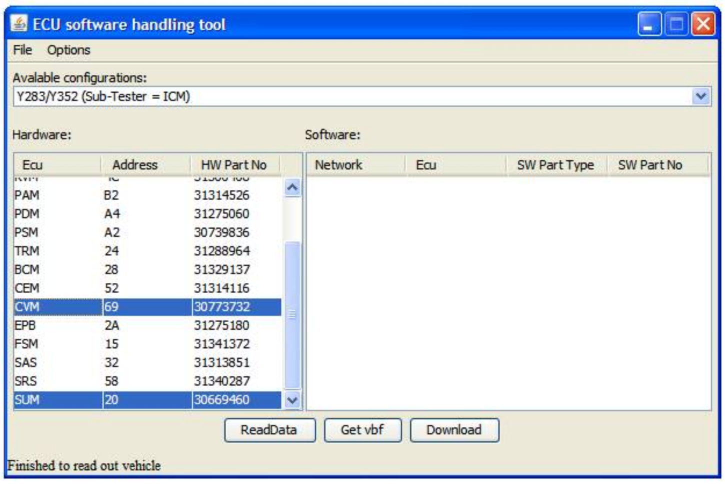
Figure C.2: Using the drop down list the user chooses the configuration (platform) the test object belongs to.



J2543Driver: Kvaser LapCAN card

Kvaser LapCAN card
Vector CANCardX
Vector CANCardXL
Vector CANCaseXL
VCT2000

Figure C.3: Using "Communication settings" dialogue, the user can choose the driver for communication with the test object. The dialogue is called via menu "Settings"



Avalable configurations: Y283/Y352 (Sub-Tester = ICM)

Hardware:

Ecu	Address	HW Part No
PAM	B2	31314526
PDM	A4	31275060
PSM	A2	30739836
TRM	24	31288964
BCM	28	31329137
CEM	52	31314116
CVM	69	30773732
EPB	2A	31275180
FSM	15	31341372
SAS	32	31313851
SRS	58	31340287
SUM	20	30669460

Software:

Network	Ecu	SW Part Type	SW Part No

Finished to read out vehicle

Figure C.4: Pressing the button ReadData the application starts to communicate with the test object. As the result the ECUs and their hardware numbers are presented in the Hardware table. The user can analyse the vehicle configuration. For example, if all necessary ECUs are presented in the test object, if an ECU has correct hardware. To get software files, the user should choose ECUs and press "Get vbf" button. The application sends the necessary information to the PIE system.

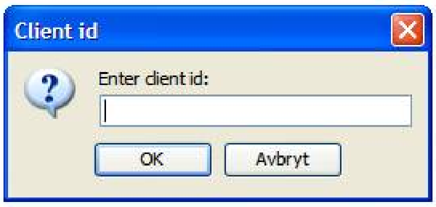
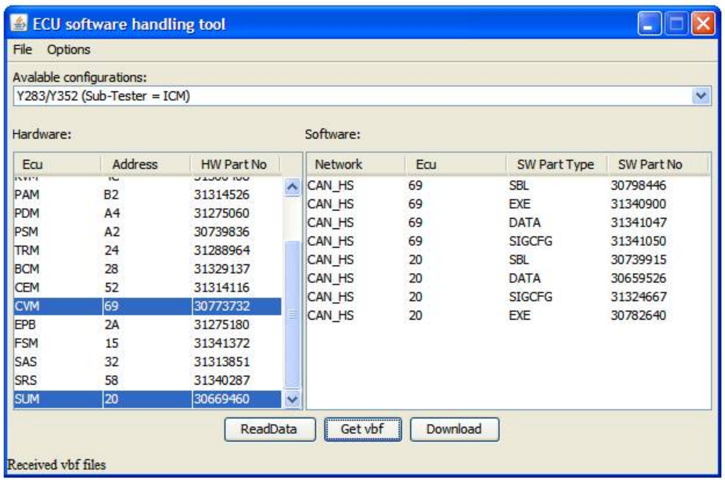


Figure C.5: The user should enter his client id.



Avalable configurations: Y283/Y352 (Sub-Tester = ICM)

Hardware:

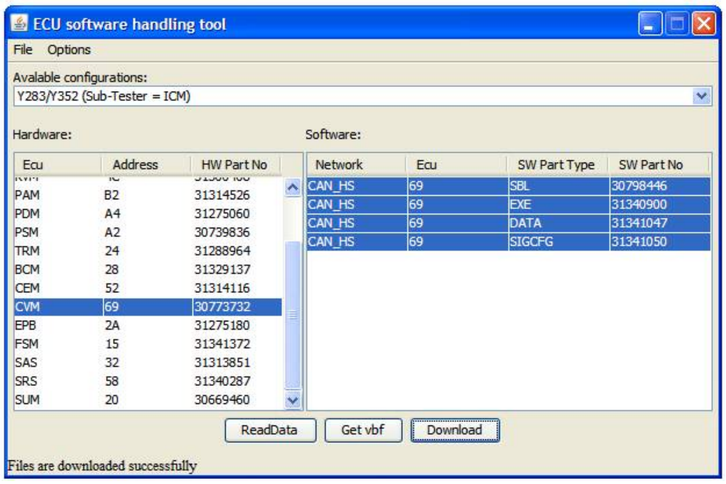
Ecu	Address	HW Part No
PAM	B2	31314526
PDM	A4	31275060
PSM	A2	30739836
TRM	24	31288964
BCM	28	31329137
CEM	52	31314116
CVM	69	30773732
EPB	2A	31275180
FSM	15	31341372
SAS	32	31313851
SRS	58	31340287
SUM	20	30669460

Software:

Network	Ecu	SW Part Type	SW Part No
CAN_HS	69	SBL	30798446
CAN_HS	69	EXE	31340900
CAN_HS	69	DATA	31341047
CAN_HS	69	SIGCFG	31341050
CAN_HS	20	SBL	30739915
CAN_HS	20	DATA	30659526
CAN_HS	20	SIGCFG	31324667
CAN_HS	20	EXE	30782640

Received vbf files

Figure C.6: The received software files are shown in the Software table.



Avalable configurations:
Y283/Y352 (Sub-Tester = ICM)

Hardware:

Ecu	Address	HW Part No
PAM	B2	31314526
PDM	A4	31275060
PSM	A2	30739836
TRM	24	31288964
BCM	28	31329137
CEM	52	31314116
CVM	69	30773732
EPB	2A	31275180
FSM	15	31341372
SAS	32	31313851
SRS	58	31340287
SUM	20	30669460

Software:

Network	Ecu	SW Part Type	SW Part No
CAN_HS	69	SBL	30798446
CAN_HS	69	EXE	31340900
CAN_HS	69	DATA	31341047
CAN_HS	69	SIGCFG	31341050

Files are downloaded successfully

Figure C.7: The user have the opportunity to download software files directly or to save them locally. To download file the user should choose one or several files and press “Download”. ”Save” function is available via menu File -> Save