Conservation and recreation of development and build environment for embedded systems

Andreas Heffsten

Computer Science and Engineering, master's level
2019

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Abstract

Today new technology is rapidly being developed, therefore it becomes difficult for developers to continue development and recreate builds of software written just a few years ago. Because of this, a quality attribute to make it easier for developers to recreate and continue development of old systems is needed. The aim of this thesis is to define what sustainable system development is and develop a quality attribute for it. The quality attribute is presented together with tactics, general scenarios and patterns that can be used to implement the quality attribute on different systems. This is to make it easier for developers to recreate and continue development of old systems.

To define sustainable system development a number of interviews were done. These interviews were done with people that are developing software and with people that are working in the area of improving systems reproducability. From these interviews sustainable system development was concluded to be defined by how good reproducibility, testability, modifiability and portability the system has.

To prove that the concept for the quality attribute works, the quality attribute was applied on the system for the ECU Coordinator 8 at Scania. With the aim to implement sustainable system development on that system. The implementation improved the reproducibility and testability of the system. But it didn’t improve the modifiability and portability, which means that the implementation at Scania can still be further improved. Some tests were performed where a developer at Scania tried to recreate the system after the implementation of the quality attribute. These tests were successful and an interview with the developer was done after the test, where the developer thought that the implementation had a positive effect on sustainable system development. From this the proof of concept for the quality attribute can be seen as successful.
## Contents

1 **Introduction** | 1  
1.1 Background | 1  
1.2 Motivation | 1  
1.3 Problem definition | 2  
1.4 Delimitations | 2  
1.5 Thesis structure | 2  

2 **Related work** | 3  
2.1 Long term digital preservation | 3  
2.2 Reproduction of software environments | 4  
2.3 Current solutions for sustainable system development | 4  

3 **Theory** | 6  
3.1 Sustainable system development | 6  
3.1.1 Definition | 6  
3.1.2 Areas of consideration | 6  
3.2 Quality attribute for sustainable system development | 7  
3.2.1 Quality attribute scenarios | 7  
3.2.2 Tactics | 8  
3.2.3 Patterns | 12  

4 **Implementation** | 16  
4.1 Coordinator S | 16  
4.2 Environment for Coordinator S | 16  
4.2.1 Tools | 16  
4.2.2 Tool importance | 18  
4.3 System scenarios | 18  
4.3.1 Recreate build environment | 18  
4.3.2 Change compiler | 19  
4.3.3 Set up environment in a new network | 20  
4.3.4 Set up environment on new hardware | 21  
4.3.5 Build program | 21  
4.4 Tactics | 21  
4.4.1 Emulation (data-centric) preservation | 22  
4.4.2 Simulation & emulation | 22  
4.5 Patterns | 22  
4.5.1 Archived environment | 22  
4.5.2 Reproducible build | 23  
4.5.3 Bill of materials | 23  

5 **Evaluation** | 24
Nomenclature

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOM</td>
<td>Bill Of Material</td>
</tr>
<tr>
<td>COO8</td>
<td>Coordinator 8</td>
</tr>
<tr>
<td>COMP</td>
<td>Common Platform</td>
</tr>
<tr>
<td>DTC</td>
<td>Diagnostic Trouble Codes</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>FC</td>
<td>File Compare</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communications Technology</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>KWP</td>
<td>Key Word Protocol</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
</tbody>
</table>
1 Introduction

These days technology is rapidly improving, which causes software and hardware to be updated or replaced often. This makes it hard to rebuild systems that were written just some years ago, since the hardware and software needed might not be available anymore. This becomes a problem when an old system becomes impossible to reproduced and the whole system might have to be recreated from scratch with new hardware and software. This is costly for companies and therefore creating a design philosophy, a quality attribute, for achieving a sustainable system development is needed.

The aim of sustainable system developments is to make sure that systems are possible to reproduce in the future. Because of that the quality attribute for sustainable system development focuses more on how the development/build environment for the system is handled, rather than how the code of the system is structured and written, since the development/build environment have bigger impact on whether the system can be reproduced or not.

1.1 Background

Scania has made commitments to customers to provide spare parts for their products for over 15 years, which makes it important for Scania that old software can be reproduced. Implementing sustainable system development on Scania’s system is a good way to achieve this. Scania has already started to implement sustainable system development in some regards, what Scania has done up to now is that they are saving the code and as many build tools as possible together in a configuration management system called Perforce. This makes the build tools available to be fetched from Perforce when a system is to be recreated. This is a good way to make sure that all build tools for each version of the code is available and easy to find. But it only works as long as the operating system and hardware is unchanged. Another problem is that the environment is locked to Scania’s network environment because of external dependencies such as license servers etc. A more complete solution needs to be found to make the system more robust, portable and easier to work with for the developer over time.

1.2 Motivation

Companies wants to be able to recreate old systems in a reliable and easy way when a change or update is needed, because this saves a lot of work and time for them. Therefore, this thesis focus on gathering recommendations, best practices, and design choices into a quality attribute for sustainable system development. Using this quality attribute will help make sure that old systems can be recreated in a sustainable fashion.
1.3 Problem definition

- What techniques can be used to achieve sustainable system development?
- How can sustainable system development be implemented at Scania?

1.4 Delimitations

The quality attribute for sustainable development does not include guidelines for writing code. The sustainable system development will focus on how the whole system can be handled to achieve sustainable system development. The evaluation of the quality attribute for sustainable system development will only be performed on one of Scania’s systems due to access, availability and time.

1.5 Thesis structure

In section 2 related work to this thesis is discussed, in section 3 the theory behind the work is discussed and in section 4 it is explained how the final result was implemented a Scania. Then in section 5 the work done is evaluated, the discussion about how the problem was solved and what the results was are shown in section 6. Section 7 contains the conclusion of the work and future work is also discussed. The last section is section 8 with all the references used in this thesis.
2 Related work

This chapter looks into previous research in the areas of long term digital preservation and reproduction of software environments, these two areas are looked into since they are important parts of achieving sustainable system development. A reflection on how companies currently are handling sustainable system development is also conducted, this reflection is based on interviews with Volvo cars and Mequet critical care.

This reflection and research are done to find out how far current solutions have come and to get some idea on how sustainable system development can be achieved better in the future.

2.1 Long term digital preservation

Some researches in the area of long term digital preservation are looked at and some different solutions to these researches are discussed below.

One way to improve long term digital preservation is to preserve the software environment as a virtual or emulated image which later can be used to emulate or virtualize the software environment on new hardware. Pros, cons and limitations for the different approaches to achieve long term digital preservation are discussed and compared in a report, to give a way on how to choose which one of the solutions are best for different software environment that is to be preserved [18]. The topic is also covered in another report but it focuses more on how virtualization can be used on analysis software environments for achieving long term digital preservation. In this report it’s discussed how long term digital preservation can be achieved by the use of CERN virtual machine(CernVM) and a HTTP-based filesystem, called CernVM file system(CVMFS) [19]. Using emulation or virtualization for long term preservation in the case for sustainable system development is a good fit. Because many tools in software development environments are system and hardware dependent, both preservation strategies helps with these problems. But they also have some other pros and cons which makes them more or less suitable depending on the software environment, which of them that are better or worse in different circumstances will be looked at later in this thesis.

Giving the developers a better understanding of the pros and cons of different approaches to implement long term preservation is important when wanting to implement long term digital preservation. To help developers, a framework for long digital preservation can be used by the developers during the development of a system, the creation of such framework is discussed in a report from the university of Edinburgh [15]. The solution of creating a framework as presented in the report is similar to sustainable system development presented in this thesis, in the sense that both aims to help developers with when and how digital artifacts should be preserved, by providing strategies that can be used
for achieving this goal.

2.2 Reproduction of software environments

Some research in the area of reproduction of software environments are looked at and some different solutions the researches are discussed below.

One solution for being able to reproduce software environments is by creating a software configuration management pattern language [14]. The pattern language in the report can be applied on many different types of software environments, this is accomplished by having multiple patterns that can be used depending on the type of software environment. Presenting different patterns to help make software environments reproducible is something sustainable system development also aims to do, but the patterns in sustainable system development will be for the developers and focus on software development environments. This will make the patterns in sustainable system development more narrow and more focused on a specific area then in the report.

2.3 Current solutions for sustainable system development

To find out what current solutions that companies are using today for achieving sustainable system development, people from Volvo cars and Mequet critical care were interviewed about what solutions they are using in their respective companies.

From the interview with Volvo cars it was discussed that Volvo cars aim to always be able to recreate old environments, rather than continuously updating them to keep them running. This is approached by documentation the environments and the requirements the environments have, so the environments can be recreated from the documentation later if needed. To make it easier to recreate environments Volvo Cars also strives to have environments without external dependencies. But this is not always possible and therefore some external dependencies may still exist in some of their environments. The build tools used in the environments are saved together with the source code to make sure they are available later and easy to find. In order to verify recreated environments Volvo cars saves binaries made in the original software environment with the source code, to have as reference that can be compared with new binaries created from the reproduced environment with the same code. [6]

Creating documentation of the environments is a good way to make it possible to recreate environments later on, but it requires old hardware and software to be available for the company. Striving to have no external dependencies is a good way to make it easier to recreate environments, since it is possible to control the whole environment. Using a reference binary file to compare new builds from a recreated environment is a good way to verify that it works as its supposed to. When considering everything Volvo cars does, it is possible to say
that they have come quite far in approaching sustainable system development in most areas. The only problem that needs to be solved is that reproduction of the environment only works as long as appropriate hardware for the environments still exists and are available for the company.

Mequet critical care is using some different approaches to handle the problem in this thesis. Mequet critical care documents all tools used in the environments in a so called Bill Of Materials (BOM) and recreates their environments by the use of the BOM. Then recreated environments are made sure to be working as expected by the use of a test setup. 5

Using a BOM helps to find the tools which are needed to recreate the environments and by being able to test the whole environments with a test setup makes it possible to verify that it works. But the setup also needs to be preserved and verified in some way or made sure that the setup is always available and that it doesn’t change over time. Using these techniques approaches sustainable system development in the area of preservation, to some degree, and verification, though it can be hard to maintain since the setup in itself is an software environment. but lacks in the area of being able to recreate the environments, since even if you know which tools exists in the environments from the BOM, how to make sure that they are available to be used and how do you make sure that they have hardware that can run them.
3 Theory

This chapter will cover the theory for sustainable system development. Some patterns and tactics that theoretically can achieve sustainable system development will be covered and included in the sustainable system development quality attribute.

3.1 Sustainable system development

Sustainable system development is used to improve the reproducability of a system in the future. Software system differs widely from each other both in size and complexity therefore there is no ”one size fits all” approach for achieving this. Rather many different patterns, tactics and techniques for the different tactics will be needed to cover the many different types of software environments existing today. It’s important to keep in mind that the patterns, tactics and techniques presented in this thesis are not the only possibilities to implement sustainable system development, these are just the ways which were found during the work with this thesis. There are most certainly many more patterns, tactics and techniques which can be added to give more ways how sustainable system development can be implemented.

The patterns and tactics in this chapter tries to cover four different problem areas for sustainable system development, these are reproducability, testability, modifiability and portability. Improvements in these areas are to make the recreation of the system easier in the future. Depending on the software environment the areas have to be solved to different degrees. To which degree each of the areas needs to addressed will lead to what patterns and tactics that are best suited to be used for implementing sustainable system development on that system.

3.1.1 Definition

The definition for sustainable system development is that during development of a software system incorporate the creation of a way to make it possible to recreate the software system in the future, without having to make any big changes to the software system in question.

3.1.2 Areas of consideration

When wanting to implement sustainable system development there are four problem areas for the system that needs to be considered, as mentioned earlier. The four areas are reproducibility, testability, modifiability and portability. The meaning and importance each of the areas have for sustainable system development will be explained briefly below.
Testability
Testability for a software is the ability to be able to verify the correctness of the software. This is important for sustainable system development since it must be possible to verify that a recreated system works correctly.

Modifiability
The modifiability of a software system is how easy or hard it is to modify and change it. Having good modifiability makes it easier to make changes to the system. Which is good, since some parts might needed to be modified to be able to recreate the system in the future.

Portability
Portability of a software is the possibility to use the software in different environments. Being able to move the software environment (all tools used etc) to a new environment can be important since the old environment might not be available anymore when wanting to recreate the software environment.

Reproducibility
Reproducibility is the ability of being able to recreate something. In the case for sustainable system development, an software environment. This is the main problem that sustainable system development tries to solve.

3.2 Quality attribute for sustainable system development
The quality attribute for sustainable system development is a non-functional requirement that can be set on a system to achieve sustainable system development. The quality attribute in this thesis is presented in a similar way as in the book *Software Architecture in Practice* [17], with scenarios, tactics and patterns that can be used to achieve the quality attribute requirements in a system.

3.2.1 Quality attribute scenarios
A quality attribute scenario is a quality-attribute-specific requirement. A scenario consists of 6 parts: [17]

1. Source of stimulus -human, computer system, etc
2. Stimulus -a condition that needs to be considered
3. Environment -what are the conditions when the stimulus occurs?
4. Artifact -what elements of the system are stimulated
5. Response -the activity undertaken after arrival of the stimulus
Table 1: Explanation of the words used in the quality attribute scenarios.

<table>
<thead>
<tr>
<th>Word</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>The entity or source that initiates the stimulus.</td>
</tr>
<tr>
<td>Stimulus</td>
<td>The condition that need to be considered when it arrives at the system.</td>
</tr>
<tr>
<td>Artifact</td>
<td>The part(s) of the system that is being affected by the stimulus.</td>
</tr>
<tr>
<td>Environment</td>
<td>The condition the system is in when the stimulus occurs.</td>
</tr>
<tr>
<td>Response</td>
<td>How the system reacts to the stimulus.</td>
</tr>
<tr>
<td>Response measure</td>
<td>How the response can be measured.</td>
</tr>
</tbody>
</table>

6. Response measure - when the response occurs it should be measurable so that the requirement can be tested

An explanation of the parts can be seen in Table 1. In order to create a framework for creating system specific quality attribute scenarios, there are some general quality attribute scenarios presented with the quality attribute for sustainable system development. These general quality attribute scenarios are shown in Table 2. The general scenarios were created from discussions together with people at Scania, that are working with Scania’s systems, on what possible scenarios they saw for the quality attribute. This framework for creating system specific quality makes it easier for people to create quality attribute scenarios for just their system later, an example of a quality attribute scenario can be seen in Figure 1. The system specific system scenarios are then later used as verification that the quality attribute for sustainable system development have been implemented in an desirable way on a specific system.

![Figure 1: Example quality attribute scenario for sustainability](image)

3.2.2 Tactics

Quality attribute tactics are used to control the stimulus, in order to get the desired response from the system as illustrated in Figure 2. The main tactics for achieving the quality attribute requirements in this thesis are: prevent ripple
Table 2: General quality attribute scenario for sustainability.

<table>
<thead>
<tr>
<th>Portion of scenario</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>developer, system administrator, provider, audit controller</td>
</tr>
<tr>
<td>Stimulus</td>
<td>change OS/tools, add tools, modify code, reproduce environment, OS/tool not provided anymore, server moved/removed, move development environment, build program, continue development (make new branch)</td>
</tr>
<tr>
<td>Artifact</td>
<td>OS, tool, external server, code, development environment</td>
</tr>
<tr>
<td>Environment</td>
<td>after delivery, reproduced environment</td>
</tr>
<tr>
<td>Response</td>
<td>make change without affection how the environment build the program, backup tool/OS can replace the tool/OS that’s not available anymore and environment can be used as before, produce an environment with same build functionality as when the program was last built, moved environment functions the same way as before the move, removal/move of server doesn’t affect the environment, program is built and is equivalent to previous builds of same version</td>
</tr>
<tr>
<td>Response measurement</td>
<td>time, cost, effort</td>
</tr>
</tbody>
</table>

effects, reduce external dependencies and preservation. The tactics and the different techniques that can be used by each of the tactic are showed in figure 3. These three main tactics will be explained in more detail below.

![Figure 2: General use of tactics for quality attributes.](image)
Prevent ripple effects
The prevent ripple effect tactic is most effective when the system have many tools that have a high possibility to be changed or updated in the future. Since the aim of the tactic is to reduce the amount of tools affected when another tool is changed or replaced in the environment. The aim of this tactic is to make it easier to make changes in the environment, since some changes might be needed to be done to recreate the environment later. The techniques that can be used to achieve this tactic are:

- Keep interface - Always have same communication interfaces between tools.
- Use intermediary - Use an intermediary between the tools that handles the communication.

Reduce external dependencies
The reduce external dependencies tactic is good to use when the environment has some external dependencies, that makes it difficult to preserve the environment in a good way. The techniques that can be used to achieve this tactic are:
• Internalization - Create internal dependencies from the external dependencies.

• Abstraction - Abstract technical requirements, such that equivalent replacements can be found for the external dependencies.

• Emulation & simulation - Emulate/simulate external dependencies.

Of these techniques, internalization is often the most effective technique to use when having an external dependency, since it completely removes the external dependencies by incorporating it inside the environment. The abstraction technique is used by abstracting the technical requirements of the external dependencies, such that an equivalent replacements can be identified and applied in the future. The last technique “emulation & simulation” is applied by emulating the whole external dependencies so that it can be preserved together with the environment. This is a good way to preserve the external dependencies, but using emulated environment takes up more memory and have worse performance than a non emulated environment.

**Preservation**

The preservation tactic is most effective when having a software environment that is hard/not suitable to keep alive for a long time. The goal of the preservation tactic is to make it easier to recreate the system later by preserving it in different ways. The techniques that can be used to achieve this tactic are:

• Echnical preservation (techno-centric) - Preserve original hardware and software in same state.

• Emulation (data-centric) - Emulate original hardware and operating environment, keeping software in same state.

• Migration (functionality-centric) - Update software as required to maintain same functionality, porting/transferring before platform obsolescence.

• Cultivation (process-centric) - Keep software ‘alive’ by moving to more open development model, by bringing on board additional contributors and spreading knowledge of process.

• Hibernation (knowledge-centric) - Preserve the knowledge of how to resuscitate/recreate the exact functionality of the software at a later date.

• Deprecation - Formally retire the software without leaving the option of resuscitation/recreation.

By observing the number of different techniques above, it is possible to conclude that this tactic is achieved by preserving the system in many different ways. One can either preserve an environment by using an archive and save the whole environment there, preserve a documentation of the complete environment works
and are set up or update the environment as times goes on. Using an archive have a big one time cost when the environment is retried and archived. While updating the environment as time goes on will have a continuously cost that will become large if the system have to be alive for a long time. By updating the environment, it is easier to make sure that the environment will work in the future. While it is not possible to be 100% sure that an archived solution will work in the future. For example if the environment is preserved by using an emulation image, then an emulator is needed that both can be used on the new hardware and can run the image that is preserved, that such emulator will be available in the future is impossible to know for certain.

### 3.2.3 Patterns

A pattern is used to improve different aspects of the quality attribute. The pattern is either one tactic or a combination of several tactics that alone or together improves the aspect that the pattern aims to improve.

Most of the patterns in this thesis have either been inspired from the patterns in a report about how to reproduce software systems. There are four patterns in this report that can be used for improving aspects of the sustainable system development quality attribute. These first three patterns are,

- Bill of materials,
- Reproducible build,
- Version-Controlled environment.

The last pattern in the report which improves an aspect of the sustainable system development quality attribute is the ”Archived environment” pattern. This pattern could be improved with more modern technology since the report for the patterns is from 1999. So a new ”Archived environment” pattern that uses more modern technology is created in this thesis by using the original ”Archived environment” pattern in the report as an inspiration. This ”new” ”Archived environment” pattern is presented in this thesis together with the three previous mentioned patterns, as patterns for the sustainable system development quality attribute. How these patterns are related and build on each other is illustrated in figure 4.
Bill of materials
The Bill of materials pattern focus on improving the reproducibility aspect of the system.

To implement this pattern, a document containing all the components that contributed to the build of the software is needed, i.e. a BOM. The BOM also documents where the components can be found and how they are assembled together to produce the software environment. It is important to document how the components are used in the environment, since the person recreating the environment later might not have an idea how the tools are supposed to be used otherwise.

The BOM may contain the names, versions, and directory paths of operating systems, libraries, compilers, linkers, make-files, build scripts, etc. The BOM may be manually created, but generally many configuration management tools generate it as a by-product of the build. Since the BOM is also a file, it should be placed under version control and associated with the revision of the version-
controlled components that the BOM documents. This can be done applying
the same label to the version of the BOM that was used to identify the version
of source code.

It’s helpful to have documented where all components for the environment can
be found, how they are used and how they are set up together to create the
environment, in a single place. Since having all the information in one place
will make it easier and faster to find all the components needed. It will also
be easier to get an understanding on how the components are used together,
when it’s explained in one file rather than in multiple files which describes how
each component should work by them self. As a result form this, it will be easier
for someone new to recreate the environment later.

Reproducible build
The reproducible build pattern aims to improve the testability of a recreated
system.

To implement this pattern, you need to test both the build process and the
bill of materials, if a BOM is used, by producing a build from them and checking
for differences between the initial build and the process-generated build.
The simplest method would be looking at the final file size of the executable(s).
Commands like ‘cmp’ (Unix) can do a binary difference between the executables
to show if there is something lacking in the build process. Another Unix tool
is ‘spiff’, a tool that looks for embedded information (e.g, time stamps). If the
build process uses compilers that can do some nifty optimizing parallel stuff, you
may never be able to count on the exact same sequential output ordering twice.
In this case, running a regression test suite can prove that the executables are
equivalent.

Producing a reference build and preserving it for later when the environment
have to be recreated gives a way to test that a recreated environment works as
intended. Being able to verify a recreated environment through testing will make
people have more faith in that the environment works exactly as the original
system, this also improves the testability of the system.

Version-Controlled environment
The version-controlled environment pattern improves reproducibility and mod-
ifiability for the system.

To implement this pattern, identify the environment components that have an
impact on the software system and place these components under version con-
trol. Associate versions of the environment components with other components
(e.g. source code) that contribute to a build, perhaps by using a common label.
The version-controlled environment pattern makes it easier to recreate the system since all components needed are always available. It also makes it easier to modify and create new environments because of the version control.

**Archived environment**
The archived environment pattern improves the reproducability and portability for the system.

To implement this pattern, create a copy of the physical machine, that is being used to develop/compile the system, into a virtual machine image. The image should contain the same operating system, libraries, software tools and have the same processor structure as the physical machine. When the image have been created, archive it in a database together with the source code for the software and use labels that associate with that release of software, so it’s easy to find later.

This pattern makes it much easier to move the environment, since the environment is already setup inside the virtual machine image, therefore it doesn’t need to be setup manually each time. The only thing needed to get the environment up and running in a new location is a tool that can run the virtual image(s). This improves the portability of the system. It does also improves the reproducability of the system since the chances of human errors are removed when you don’t have to manually set up the environment.
4 Implementation

This section will go through how sustainable system development was implemented for the Coordinator 8, COO8, software at Scania. It will present the environment for COO8, the system specific scenarios and how tactics and patterns were used to implement sustainable system development.

4.1 Coordinator 8

COO8 is an electronic control unit (ECU), developed by Scania which are used in the truck for redirecting the communication between the other ECUs in the truck and handle the inputs from the cab as buttons, sensors etc.

4.2 Environment for Coordinator 8

The section will give describe how the development and build environment looks in the case for COO8 at Scania. It will also identifying which parts of the environment is essential for the development/building the software and which parts are more ”good to have” parts.

4.2.1 Tools

This section is about the tools that are used in the build and development environment for COO8. What purpose they have in the environment, how they are connected to each other and how they are connected to things outside of the environment.

Canalyzer
Canalyzer is a analyzing tool for bus communication, it can detect what type of communication that is occurring on the bus. The tool can both send data and log the bus communication [1]. Having Canalyzer as part of the environment creates dependency to specific hardware for the enviroment, that is needed for running Canalyzer.

CMake
CMake is an open-source, cross-platform family of tools designed to build, test and package software [2].

CUnit
CUnit is an open-source lightweight system for writing, administering, and running unit tests in C [3].

DIAB
Compiler made by WindRiver that is used on Scania for all developed system that uses 32-bit processors [4]. DIAB is licensed and get the license from a license server in Scanias network. This adds an external dependency for the
software environment.

**DIMA Tool**
This is a tool developed by Scania that generates code and documentation files. It is mainly used for handling Diagnostic Trouble Codes (DTC).

**Fenix**
Fenix is a self developed tool by Scania that from databases (JSON-files) and .tt files generates C code for the software.

**Matlab & Simulink**
MATLAB is a numerical computing environment [7], while Simulink is graphical programming environment for modeling, simulating and analyzing dynamic systems [11]. Scania uses Matlab and Simulink to build models, then Matlab are also used for generating C-code from the models. Matlab and Simulink are licensed software. At Scania the licenses for Matlab and Simulink are retrieved from a license server on Scania’s network, this adds an external dependency for the software environment.

**PCLint**
Statistic and code analyzer that can be used together with Visual studio [8].

**Perforce**
A configure management program made by Perforce Software [9], which is used for handling different of the COO8 software versions at Scania. It is located on an external server at Scania, as for Matlab this adds an external dependency for the software environment.

** PVCSCB**
A tool which is used for compiling Common platform (COMP), which are the low level code on the ECU.

**QAC**
QAC is a advanced code analyzer of C-code, it can be used together with Visual studio [10]. It needs a license and at Scania this license is stored locally on each computer.

**Visual studio 2013/2012**
Visual studio is used as a code editor and used for compiling an emulator of COO8. It is also needed for generating C code from Simulink models, but here visual studio 2012 is used. Visual studio 2013 is an integrated development environment, IDE, made by Microsoft, which require a license when used [13]. At Scania the license for Visual Studio is placed locally at each computer so there are no external dependencies for this.

**Vision**
Integrated calibration and data acquisition tool that collects signals from the ECU and external sources, measures relationships between inputs and outputs, enables real-time calibration and modification of closed loop control systems, time aligns and analyzes all information, manages calibration data changes, and programs the ECU. To use Vision a license is needed, at Scania this license is located on a USB dongle. Which creates a decency to that specific hardware is needed for the software environment.

**XCOM**
A keyword protocol identifier (KWP) /UDS diagnose client for communicating with ECUs that is developed by Scania. It aims to support all Scania developed powertrain ECUs and all CommonPlatform based ECUs.

### 4.2.2 Tool importance

Not all tools present in previous section are needed to be able to build and develop COO8. Some of them are tools for running bug checks and diagnostic. These tools can be removed without affecting the build, but it will make the development harder.

The critically for the tools in the case of build the COO8 software can be seen in table 3. The table also show if the tool are depending on a license or not and if it adds any external dependencies to the environment, both these aspects are undesirable since they make it harder to preserve the tool. By taking these parameters in to account it was deiced to not preserve Vison and Canalyzer since they both are non critical tool that requires a licenses and specific hardware to work.

### 4.3 System scenarios

In this section the system specific scenarios for Scania are presented. These system specific scenarios is then used as a reference for determining if the quality attribute for sustainable system development have been achieved on Scania's system. The scenarios were constructed after discussions together with the people that are working with Scania's development and build environment. About what they want to achieve and what can be achieved by implementing sustainable system development. This was done by using the general system scenarios in table 2 in section 3.2.1 as reference.

#### 4.3.1 Recreate build environment

The scenario is for a developer to recreate the build environment and be able to verify that the environment works equivalent to the original environment. The people at Scania though it was desirable to be able to complete this scenario within 16 hours, the scenario is presented in figure 5.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Critical</th>
<th>Licensed</th>
<th>External dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIAB</td>
<td>X</td>
<td>X</td>
<td>Server</td>
</tr>
<tr>
<td>DIMA tool</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmake</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fenix</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ninja</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matlab &amp; Simulink</td>
<td>X</td>
<td>X</td>
<td>Server</td>
</tr>
<tr>
<td>Visual studio 2013</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>QAC</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CUnit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vision</td>
<td></td>
<td>X</td>
<td>Hardware</td>
</tr>
<tr>
<td>PCLint</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PVSCB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perforce</td>
<td></td>
<td></td>
<td>Server</td>
</tr>
<tr>
<td>Canalyzer</td>
<td></td>
<td>X</td>
<td>Hardware</td>
</tr>
<tr>
<td>XCOM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Tool information

Figure 5: System specific scenario for recreating the environment at Scania

### 4.3.2 Change compiler

The scenario is to change the compiler used in the environment. It is desirable that at most two other tools in the environments have to be changed as cause of using a new compiler. This scenario is presented in figure 6.
4.3.3 Set up environment in a new network

The scenario is for a developer to recreate the environment in a new network. The people that was discussed with at Scania thought it was desirable to only have to fix at most two external dependencies for the environment to get the environment to work, this scenario is presented in figure 7.
4.3.4 Set up environment on new hardware

The scenario is for a developer to recreate the environment on a new hardware device. This is desirable to be possible to do within 16 hours, this scenario is presented in figure 8.

![Figure 8: System specific scenario for setting up Scania's environment on new hardware](image)

4.3.5 Build program

The last scenario is for a developer to be able to build the software program inside a recreated environment. This scenario is acceptable if the built program can be verified to have been built correctly, this scenario is presented in figure 9.

![Figure 9: System specific scenario for building a program in a reproduced environment](image)

4.4 Tactics

The tactics used to implement sustainable system development on Scania are "emulation(data-centric)" preservation and "simulation & emulation" on the external dependencies. The emulation(data-centric) preservation tactic is used to preserve the data environment OS, libs and tools, of the COOS environment
and the simulation & emulation tactic is to handle external dependencies.

### 4.4.1 Emulation (data-centric) preservation

The hardware at Scania have a high likelihood of being replaced during the preservation period for a product since the preservation period are at least 15 years. Because of this, the emulation tactic was chosen to be used for preserving the environment, since it removes the dependency on having specific hardware when recreating the environment. The tactic is implemented by creating virtual image of the COO8 environment, that then later can be used in Vmware workstation to create a virtual machine of the environment. The virtual machine needs to be able to access Scanias Matlab and DIAB license servers for the environment to work.

### 4.4.2 Simulation & emulation

This tactic is used on the Matlab and DIAB license servers on Scania. Having them as virtual servers makes them easier to preserve and move, since it’s possible to create backups of the servers that can be saved in a digital archive. This would be much more troublesome and take much more resources if they were physical servers.

### 4.5 Patterns

To implement sustainable system development on the COO8 environment at Scania, the patterns archived environment, reproducible build and bill of materials were used. Archived environment was chosen since Scania already is using a digital archive for old programs, thus the archive only needed to be extended to also preserve the environment. Bill of material was chosen so it would be possible to recreate the environment from scratch without the virtual image. This leads to that everything doesn’t rely entirely on having a hypervisor available to be able to recreate the environment. The reproducible build pattern is used so that the archived environments can be verified later when they are taken out from the archive and reproduced.

#### 4.5.1 Archived environment

Since Scania already has a digital archive where the source code and builds of programs are being stored, an archived environment pattern was chosen to be implemented. To implement the pattern on Scania, a virtual image with VMware workstation used. VMware workstation is a hypervisor for computers, which is used for running a virtual machine of the environment. To implement the Archived environment pattern at Scania, the current digital archive have to be extended so that each preserved system contains VMwares virtual image of the environment, a BOM file, a built program to use for reference, the source code for the program and the programs needed in the environment. This makes it possible to recreate the environment from the BOM file together with the
saved tools if the correct hardware and OS is available. While it’s also possible to recreate the environment by creating a virtual machine from the image if needed.

4.5.2 Reproducible build

The reproducible build pattern is implemented by compiling a program in the environment that is going to be preserved. Then saving that compiled version inside the VMware virtual image that is going to Scania’s digital archive. Then when the environment is recreated, the same version of the program is compiled and compared with the compiled program that was saved in the image. This is done with the use of File Compare (FC) in Windows and if there are no differences between the files, the build environment works correctly.

4.5.3 Bill of materials

The Bill of materials pattern is used to make it possible to recreate the environment without using the virtual image. The pattern is implemented by saving the tools in the environment and a BOM file together with the source code and the virtual image in the digital archive. The BOM file lists all tools, OS and libs that is in the environment, what they do and were they can be found. It also list all external dependencies and licenses that different parts of the environment needs. The saved tools can then later be used with the information in the BOM file to recreate the environment from scratch. Which is good to be able to do if the virtual image for some reason becomes unusable.
5 Evaluation

To evaluate how well sustainable system development have been achieved at Scania, a system specific scenario specified in the previous section was performed by a developer at Scania. The performed scenario was to reproduce the environment for COO8 on a new hardware, build COO8 and verify that the environment worked as expected. The result from the test was then discussed together with the developer and compared with desired result for the system specific scenario.

The time it took for the developer to perform the scenario was six hours. Two to recreate the build environment on new hardware, then four to build and verify the COO8 software. The building of the software was successful and it was possible to verify the software by comparing the produced binary with the reference binary saved in the environment. This test correspond to the specific system scenarios in figures 5, 8 and 9 in section 4.3. From these scenarios in figures 5 and 8, the developer would be able to recreate the environment on new hardware in 16 hours. In the scenario from figure 9, the developer would be able to build the program in the reproduced environment and test that the environment build the software correctly. By comparing this with the result from the test, a sustainable system development has been achieved in regards to those scenarios.

The developers opinions were that the improvements will make it easier and more reliable to recreate the environment for COO8, compared to only save all the tools needed for the build as Scania does today. The developer thought that having a virtual image with all tools setup inside it already makes it easier to set up and it will not depend on specific OS or specific hardware. The developer also thought that it would be possible to recreate the environment for a longer time into the future and that it would be easier to preserve than before. Some concerns that the developer had was that the dependency to VMware still exist, since a VMware virtual image was used for recreating the environment. The second concern was that the modifiability might be to low when having multiple developer working with environment. The last concern that was brought up during the test was that the virtual machine might have too big performance drop-off, since the virtual was a bit slow sometimes. But in general, the developer thought that the changes that was made improved the aspects of sustainable system development for the COO8 environment at Scania.
6 Discussion

The aim of sustainable system development and its quality attribute is to make software system reproducible in the future, regardless of what type of software system it is applied on. When creating the quality attribute for sustainable system development mainly software developments systems at Scania were looked at, this might have made the quality attribute more applicable towards software development systems and some problems for other type of systems might have been missed to take into consideration.

The proof of concept made at Scania did not improve all aspects of sustainable system development in the system used for the proof of concept, so it’s not a complete proof of concept for the quality attribute for sustainable system development. During the proof of concept most of the aspects were improved, which show that implementing sustainable system developments will have a positive effect on the desired properties of the system. Therefore this makes it is possible to say that implementing the quality attribute is at least a step in the right direction for achieving sustainable system development.

6.1 Possible improvements

In the implementation at Scania a VMware workstation were used, this is a virtualization tool which means that if you have a x86 processor as a host running VMware workstation, then VMware workstation is only able to virtualize x86 hardware. Therefore by using VMware workstation some dependencies to the physical hardware will still remain. This is not good for sustainability, but this problem can be fixed by using emulation instead of virtualization. By using emulation, all dependencies to physical hardware are removed and therefore makes it possible to emulate, for example x86 hardware on a x64 host. The only problem when using emulation is that it has a bigger performance drop of than virtualization, which most of the time can achieve almost near-native performance. It was decided to use VMware workstation at Scania since it’s already being used by some departments and it have better performance than emulated environment. But since Scania have to be able to provide spare parts for over 15 years, it might be a good idea to also preserve an emulation image together with the virtualization image, which can be used where the virtualization image can’t be used. Or change to preserve the environment using only emulation since it’s more robust.

The Matlab and DIAB license servers is right now handled in Scania by having them as virtual servers. This approach do make it easier to preserve them, but it would be much easier to internalize the licenses and by doing so, removing the external dependencies. This is possible to do, but Scania needs to have an agreement with the provider of Matlab and DIAB to get licenses that are active the whole period of the time the environment is desired to be preserved for it to work.
To improve the modifiability of the system it can be a good idea to add a container tool to the virtual machine, for example Docker. Since that would make it easier to change the tools in the environment between version of the same program. The virtual machine would act as a base containing all the basic tools and the dockers would have the version specific tools. This would be much easier than having to make a new virtual machine for each version and also allow much more modification in the environment.

6.2 Problems

Some problems that was encountered during the the project was finding out how different licenses did work and how they were allowed to be handled for example, if it was allowed to run a Windows 7 virtual machine on a Windows 7 host on the same license. Another example is if it was allowed to save programs with local licenses in a virtual machine and have backup of that virtual machine saved in a digital archive. It was also hard to make any bigger changes in the software environment during the proof of concept. Because too find out if why different software artifacts that was made during the development was needed, a lot of people had to be contacted and checked with. This could take up much time since some times it was quite tricky to find the right person with the right information about the what the different artifacts were used for. This can be contributed to that Scania is a big company with a lot of employees that are working in a high tempo and therefore they are often busy.
7 Conclusions and future work

This section will discuss the results of the report, what has been done, what can be improved and what future work for the area.

7.1 Conclusion

The thesis was to define what sustainable system development is and which aspects of the system it aims to improve. The thesis also defines a quality attribute for sustainable system development, that can be used to help implement sustainable system development on different systems. During the work some tactics and patterns were found that can be used to improve the different aspects of sustainable system development on a system. These are presented as part of the quality attribute as examples/guidance of what can be done to implement sustainable system development. This also help identify which parts of the system should be considered when wanting to implement the quality attribute on a system.

The implementation of sustainable system development at Scania by using this quality attribute were successful in most aspects. It help improved the reproducibility, testability and portability in the system for COO8, the modifiability of the system wasn’t focused on much which lead it to not improving from the implementation of sustainable system development. The aspects of sustainable system development can still be improved on the system for COO8, but this first step gives a way for Scania to see that there is much to gain by achieving sustainable system development in a system.

7.2 Future work

The quality attribute for sustainable system development can always be improved with more patterns and tactics that can handle even more cases than those presented in this report. New patterns and tactics will probably have to be added or the old ones will have to be improved to handle new systems as new and better technology becomes available. New general scenarios will probably have to be added for problems that doesn’t exist with today’s systems, but might arise in future system. In order to give better help in implementing sustainable system development in the future.

An area that can be looked on in the future to help the implementation of sustainable system development, is if it is possible to improve the performance of emulated environments. Since it can be quite demanding to build software programs and with to low performance, it will become unfeasible to use emulation. Since it will take to long to develop and build software programs.

It’s possible for Scania to further improve sustainable system development aspect in their systems. Scania can buy needed licenses which are active for the
whole period it’s desired to preserve a system, to make sure that needed tools can be used in the future. It’s also possible to buy licenses that can be internalized into the virtual machines. But for this, Scania have to make agreements about the licenses with Matworks for Matlab and Windrivers for DIAB.
8 References


[18] K. R. U. of Freiburg), P. F. (Tate), and T. E. K. College. Introduction
to an emulation-based preservation strategy for software-based artworks.