LERO THE IRISH SOFTWARE ENGINEERING RESEARCH CENTRE

Experience Report on Industrial Product Derivation Practices within a Software Product Line Organisation

Pádraig O'Leary
RiSE – Reuse in Software Engineering and Computer Science Department,
Federal University of Bahia, Salvador, BA, Brazil

Steffen Thiel
Furtwangen University of Applied Sciences, Germany

Ita Richardson
Lero – The Irish Software Engineering Research Centre
University of Limerick, Ireland

4th November 2010

Contact

Address	Lero International Science Centre University of Limerick Ireland
Phone	+353 61 233799
Fax	+353 61 213036
E-Mail	info@lero.ie
Website	http://www.lero.ie/

This work is partially supported by Science Foundation Ireland
under grant no. 03/CE2/I303-1

Lero Technical Report Lero-TR-2010-03

Experience Report on Industrial Product Derivation Practices within a Software Product Line Organisation

Pádraig O'Leary, Steffen Thiel, Ita Richardson

Abstract

Inefficient product derivation practices can greatly diminish the productivity gains expected from a software product line approach. As a foundation for systematic and efficient product derivation a better understanding of the underlying activities in industrial product line development is required.

This technical report presents the main findings from a case study. It provides empirical evidence on the organisational structure, roles and responsibilities, and the derivation process of an industrial software product line company. The report adds to the body of empirical evidence on product derivation practices.

1 Introduction

A Software Product Line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. The SPL approach makes a distinction between domain engineering, where a common platform for an a number of products is designed and implemented, and application engineering, where a product is derived based on the platform components [2]. The separation into domain engineering and application engineering allows the development of software artefacts which are shared among the products within that domain. These shared artefacts become separate entities in their own right, subscribing to providing shared functionality across multiple products.

It is during application engineering that the individual products within a product line are constructed. The products are built using a number of shared software artefacts created during domain engineering. The process of creating these individual products using the platform artefacts is known as product derivation.

Product derivation is the process of constructing a product from a Software Product Line's (SPL) core assets [3]. An effective product derivation process can help to ensure that the benefits delivered through using these shared artefacts across the products within a product line is greater than the effort required to develop the shared assets. In fact, the underlying assumption in SPL that "the investments required for building the reusable assets during domain engineering are outweighed by the benefits of rapid derivation of individual products" [4] might not hold if inefficient derivation practices diminishes the expected gains.

In the context of inefficient product derivation, a number of publications speak of the difficulties associated with the process. Hotz et al. [2] describe it as "slow and error prone, even if no new development is involved". Griss [5] identifies the inherent complexity and the coordination required in the derivation process by stating that "…as a product is defined by selecting a group of features, a carefully coordinated and complicated mixture of parts of different components are involved". Therefore, the derivation of individual products from shared software assets is still a time-consuming and expensive activity in many organisations [3].

Despite this, there has been little work dedicated to the overall product derivation process. Rabiser et al. [6] claim that "guidance and support are needed to increase efficiency and to deal with the complexity of product derivation". As Deelstra et al. [3] states there "is a lack of methodological support for application engineering and, consequently, organisations fail to exploit the full benefits of software product families."

Investigation into industrial product derivation practice is considered an interpretive study. Walsham [7] stated that "case studies provide the main vehicle for research in the interpretive tradition". The case study approach has always been one of the most popular research strategies [8]. It is a powerful and flexible technique, considered suitable for exploratory research both prospectively and retrospectively [9]. A case study is especially helpful in situations where researchers are seeking to develop understandings of the dynamics of a phenomenon in its natural context [10]. It is often considered to be the optimal approach for researching practice based problems, where the aim is to represent the case authentically “in its own terms” [11].

In this report, we present a case study we conducted on industrial product derivation practices. The case study was conducted in a large automotive supplier and the observations from this study are detailed here. The observations are focussed on the organisational overview of the SPL, roles and responsibilities and the product derivation process within the company.

The report contributes to an improved understanding of industrial product derivation practice. It can be considered a source of empirical evidence for derivation practice and thus can enrich existing theory on product derivation practice.

The paper is structured as follows: Section 2 details the case study methodology. Section 3 gives an overview of the product line platform. Section 4 describes the organisational overview. Section 5 presents the main roles and responsibilities. Section 6 discusses the product derivation process within the company. Section 7 details the findings from the case study. Finally, section 8 presents the conclusion.

2 Case Study

For the case study, the researcher collected data on the product derivation process of software-intensive automotive product lines in different business units within the company. The company was chosen for the case study because previous SPL efforts within the company have been judged a success by their peers [12]. The case study was carried out in conjunction with the corporate research division.

2.1 Data Source

The researcher met with the case study company where he gave a presentation on the research project and the nature of the commitment required for any participant company. In return for company participation, on completion of the case study the researcher would organise a workshop where he would present a set of recommendations for improvements within the company's product derivation process. The researcher would introduce some alternative derivation practices as identified in the literature. On completion of the research project, the researcher would make any findings available to the participating company.

The company agreed to participate and organised for the researcher to work with the companies Corporate Research division. The Corporate Research division was interested in analysing product line product derivation methods applied in different Business Units within the organisation. The goal of the work was to strengthen the ability of Corporate Research to support and advise different Business Units in product derivation. The Business Units seeking support and advice were within the software-intensive automotive product lines division. Each Business Unit platform contained software and algorithm components, hardware modules and housing concepts. Members of two Business Units were made available to Corporate Research for the duration of the project.

Corporate Research nominated a person to liaise with the project for planning and execution of the research and four staff member from various business units were made available for the onsite visit. After discussion three primary work tasks were identified for the case study. These tasks were:

1. Identification of product derivation practices within the company
2. Evaluation of product derivation practices within the company
3. Recommendations to improve product derivation practices

A timeline was established with expected delivery dates on interim results and final delivery of recommendations agreed upon.

2.2 Data analysis procedure and methods

According to Yin [10], the use of multiple sources of evidence in a case study allows an investigator to address a broader range of historical, attitudinal, and behavioural issues. The use of multiple sources of empirical evidence provide an opportunity for triangulation in order to make any finding or conclusion of the study more convincing and accurate [10]. Yin identifies six sources of evidence:

documentation, archival records, interviews, direct-observation, participant-observation, and physical artefacts [10].

While conducting the case study the researcher had access to the following sources of evidence:

- Company documentation
- Collective group notes from workshop
- Whiteboard drawings of company practices collected from workshop
- Researcher notes of workshop discussion
- Anecdotal evidence from company employees

Prior to an onsite visit to the case study company, the researcher had access to internal company documentation. These documents included information on product derivation practices within a particular business unit, organisational structure of the company's SPL teams and information on various derivation techniques applied within the various business units. The researcher performed an initial review of company practices based on this documentation. This initial review of company practices would be used to facilitate discussion during the onsite visit.

For the onsite visit to the company, the researcher organised a two day workshop. The main theme of the workshop was product derivation practices within the case study company. The researcher examined the company product derivation practices under a number of headings:

- Activities
- Artefacts
- Stakeholders
- Tools
- Techniques
- Constraints
- Open issues

Two other researchers, one of whom had extensive experience in conducting case study research, accompanied the primary researcher. Each researcher had a specific role for the workshop; one recorded the workshop discussion through note taking and a second recorded the main points on a projected PowerPoint presentation. Note taking in this manner helped the workshop discussion in two ways. Firstly, it kept everybody involved in the discussion and kept the discussion on track. Secondly, projecting results encouraged the workshop group members to form a consensus on topics. The third researcher was responsible for the whiteboard. The whiteboard was used to illustrate company practices, and all company employees were encouraged to actively engage with the whiteboard. Each whiteboard drawing was printed before the board was wiped clean to serve as a record of the discussion. In Figure 1, a sample whiteboard drawing shows one of the steps in the company's product derivation process. This figure describes how the step 'Resolve Variation Points According to Customer Requirements' took various inputs such as the Product Software Requirements Specification (P-SRS) and produced outputs such as the Product Software Architecture.

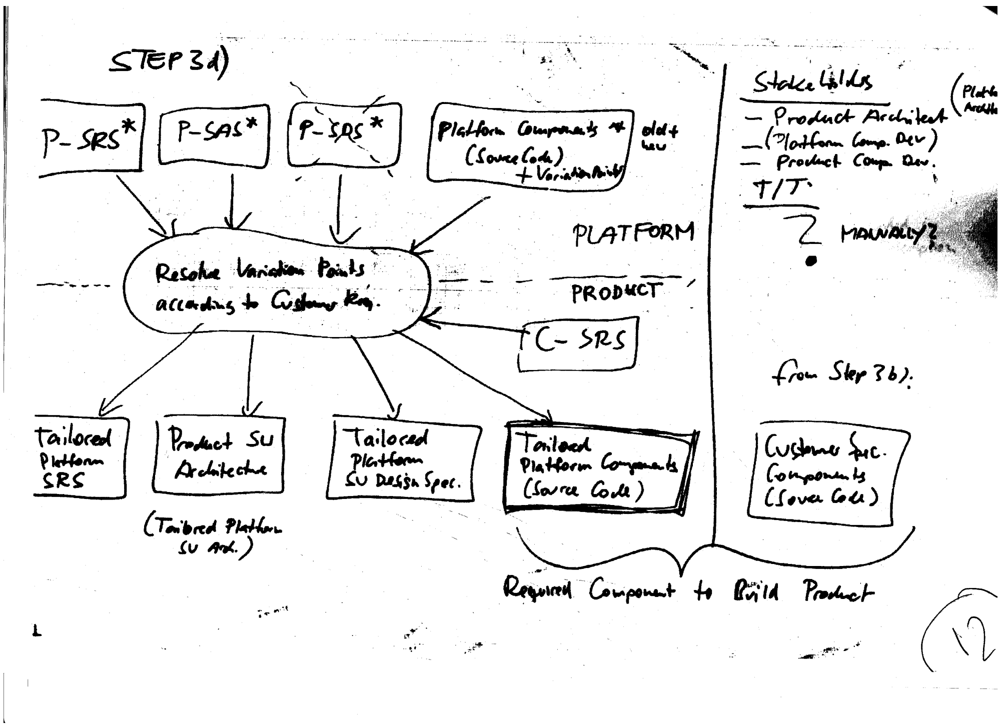

STEP 3d)
P-SRS*
P-SAS*
P-SDS*
Platform Components * (Source Code) + Variation Points
old + new
Resolve Variation Points according to Customer Req.
PLATFORM
PRODUCT
C-SRS
Tailored Platform SRS
Product SW Architecture
(Tailored Platform SW Arch.)
Tailored Platform SW Design Spec.
Tailored Platform Components (Source Code)
Stakeholders
– Product Architect
– (Platform Comp. Dev)
– Product Comp. Dev.
T/T
MANUALLY?
from Step 3b):
Customer Spec. Components (Source Code)
Required Component to Build Product
12

Figure 1 Sample Whiteboard Drawing from the Case Study Workshop

After the workshop the collected data was used to create a technical report on the company derivation practices. The technical report described the company process of deriving an individual customer product from the product line platform. The report contained information on tools used to assist in derivation projects and organisational structure and roles within those derivation projects. The researcher made recommendations for the introduction of Agile elements in the companies derivation process.

3 Platform Overview

The Platform consists of two main types of artefacts:

1. Supporting documents
    a. Specification of requirements, architecture and tests
2. Product components
    a. Software and Algorithm Components
    b. Hardware Modules
    c. Housing Concepts

The scope of the Platform (features included and excluded) is defined by introducing a number of typical product types. A product type is a typical product built from the Platform.

The modules from the platform can be taken as a starting point for designing and implementing customer-specific modules. Note: In the Platform, an "interface" is a mechanism for the implementation of variability, i.e. "open connectors" in the platform. Some "interfaces" are still open after integrating the modules of the Platform Library and require the integration of further customer-specific modules. Interfaces between the different modules of the Platform Library are not considered as "interfaces".

A customer product is never built from scratch. A customer Product is always based on existing assets (requirements, design, code etc.). Either 1) The basis for the customer Product is the platform or 2) some precursor components (from pervious projects within the platform). However, this would only happen in a special case when everything becomes more mature.

The platform defines a number of rules.

Rule 1: Platform assets (architecture, design, software and algo modules etc.) should not be changed in customer projects (what is part of the platform→)

The Platform assets are flexible to some degree using two main mechanisms: They can be configured (via variability mechanisms) or some provide interfaces and can be extended. If any platform assets have to be changed, then all the module tests have to be redone.

The platform parts of the customer project should not be changed. The customer-specific parts are realised in a subsystem. As a template for the customer-specific subsystem, a Customer Template can be used. This is one of the 'typical' product types that is used to define the product line scope.

Rule 2: Use the customer specific templates as a starting point for creating the customer-specific part of the Platform-based product.

In a customer project, the Platform Library, Testing Bench and Customer Template will be copied and taken as a starting point for the customer-specific software architecture and design. The subsystems Platform Library and Testing Bench are not changed. The subsystem Customer Template, however may be used as a starting point for customer-specific software parts and modified.

4 Organisational Overview

In this section we describe the organisational structure of the companies product line. The description mentions a number of roles, these roles are described in more detail in the following section.

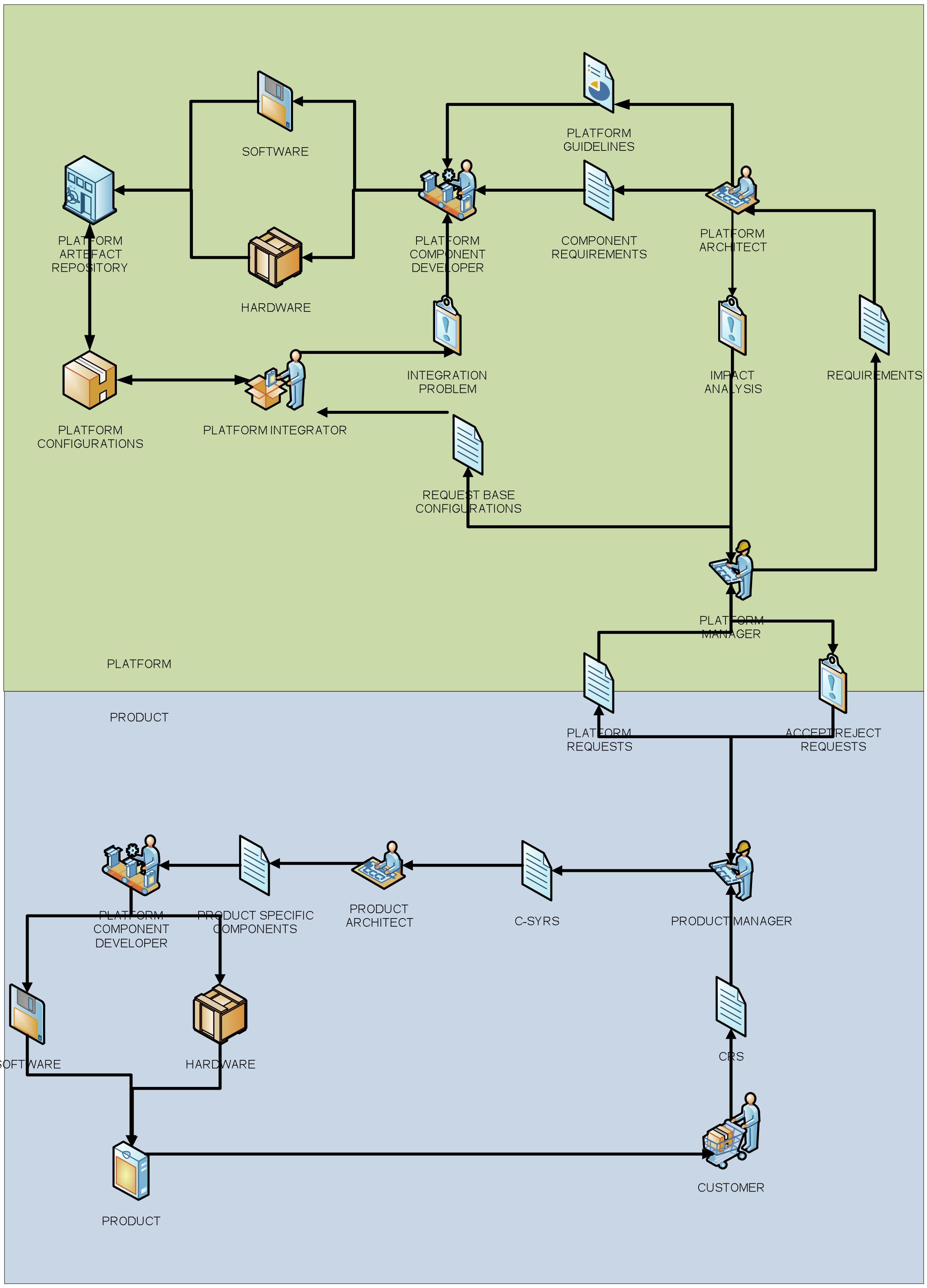


Figure 2 – Organisational Structure

4.1 Platform Team

The **Platform Manager** receives requirements from the **Product Manager**. These requirements are specific product requirements that the **Product Architect** wishes to see implemented by the platform. The **Platform Manager** (he is interface, point of contact) passes the requested requirements to the **Platform Architect**. The **Platform Architect** performs an impact analysis study on the impact to the platform if the requested platform requirements are implemented. The **Platform Architect** advises the **Platform Manager** who will accept or reject the requested platform requirements. The platform requirements, which are deemed suitable, and inline with the platform roadmap are included in the platform development plan. The Platform Architect who deems requirements as requiring either new platform component development or adaptation of existing platform components decides the implementation plan for these new requirements. Responsibility for the development/adaptation of platform components is left to specific **Platform Component Developers**.

The **Platform Integrator** is responsible for integrating the platform components and creating base platform configurations. It is the **Platform Manager** who informs the **Platform Integrator** what configurations to build and when to build them. The **Platform Integrator** is also responsible for testing the created configurations using the test plans he receives from the **Platform Architect** (In the platform the architect has a powerful role because he is responsible for the technical infrastructure. He has to coordinate the technical activities).

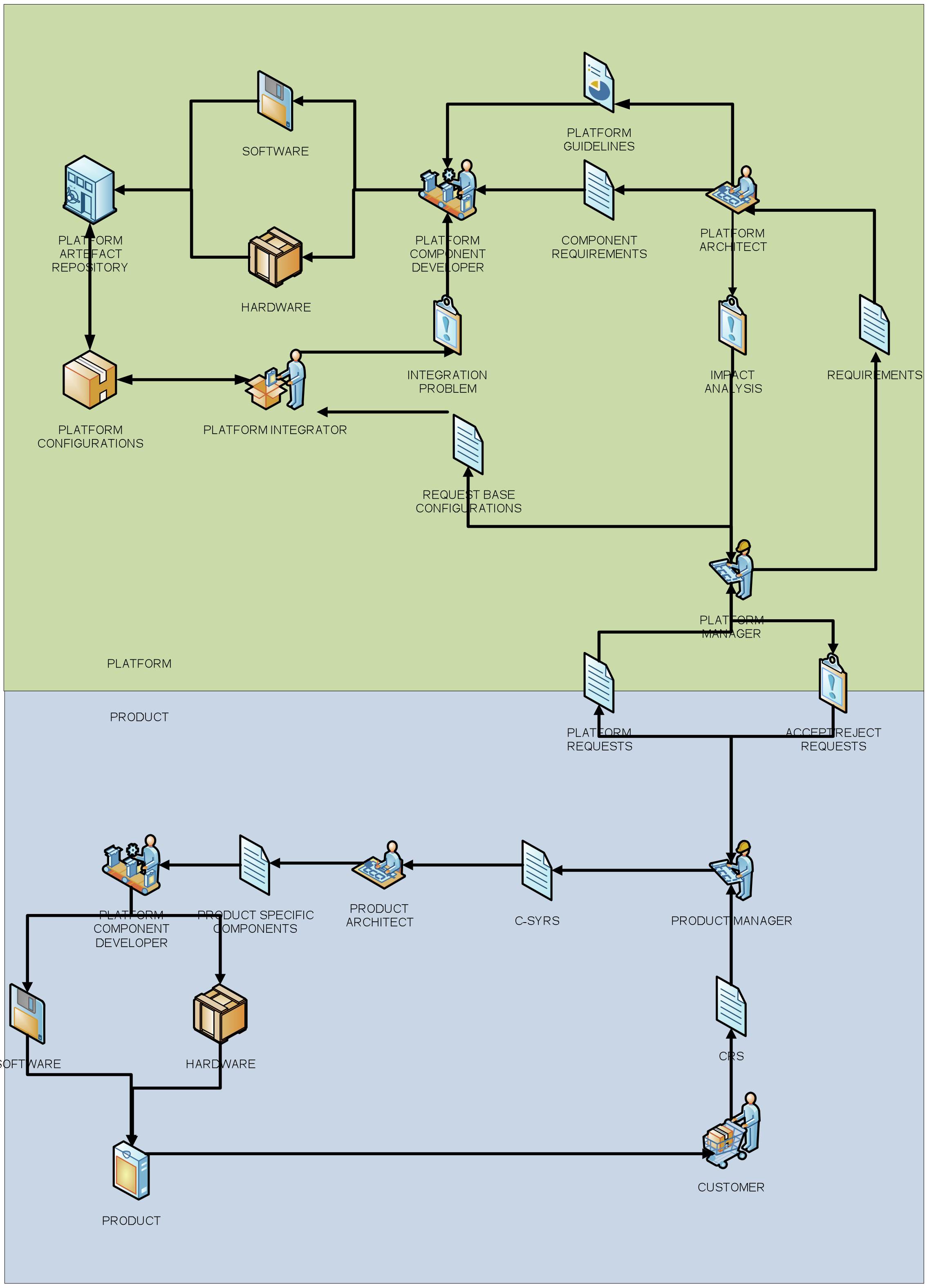
If the **Platform Integrator** has any problems integrating the platform components for a specific configuration he informs the **Platform Component Developer**. The **Platform Component Developer** who is responsible for the particular component causing the integration difficulty must adapt or change the component.

The **Platform Test Team** creates the system test plan, which they use to guide them when testing base configuration created by the **Platform Integrator**.

4.2 Product Team

Each **Product Manager** is responsible for a particular customer product, for example an airbag control system for BMW. The **Product Manager** responsibilities include: customer relationship management, negotiating customer requirements and liaising with the Product Architect regarding the final product requirements.

The **Product Architect** receives the product requirements from the **Product Manager**. The **Product Architect** plays a key role in coverage analysis asking "*What does the platform have to offer the Product Team?*". The **Product Architect** through the **Product Manager** makes specific Platform Requirements Requests to the **Platform Manager** (



). These requests are product requirements that the **Product Architect** wants to see implemented at platform level. The **Platform Manager** will either reject or accept these requirements depending if they fall under the scope of the Platform. If the requested requirements are accepted then these new platform requirements will be implemented and become accessible to the **Product Architect** in a new platform release.

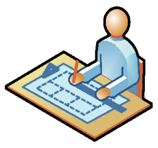
The **Product Architect** assigns product requirements to **Product Specific Component Developers**. They implement the customer specific components that satisfy specific product requirements.

5 Roles and Responsibilities

The case study company had a number of roles and associated responsibilities. These roles were:

- Platform Architect
- Platform Integrator
- Platform Component Developer
- Platform Manager
- Customer Specific Component Developer
- Product Developer
- Product Manager

5.1 Platform Architect:



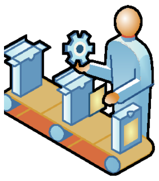
- Defines the Platform Architecture
- Ensures that guidelines of Platform Architecture are achieved by Platform Development
- Impact analysis for Platform Architecture changes
- Define System Test Plans
- Tests pre-assembled platform configurations. Partial solutions formed and integration tested

5.2 Platform Integrator

- Integrates Platform Components up to whole system (incl. base configurations)

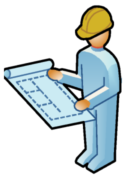
- Tests the integrated system

5.3 Platform Component Developer (SW/HW)



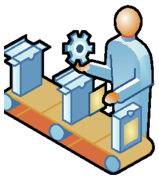
- Design and Development of Platform Assets
- Requirements engineering for Platform Assets
- Reviews and Tests of developed components
- Provide Test Plan for Product Developers

5.4 Platform Manager



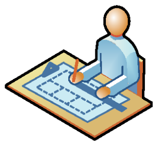
- Requirements management for Platform Assets
- Defines scope for platform
- Define Platform Configurations
- Delivery (to Product Development)
- Responsible for reusability of components (e.g., reuse degree of components)

5.5 Customer Specific Component Developer



- Development of customer specific components
- Analysis of requirements
- Analysis of platform assets
- Adaption of platform assets
- Development of customer specific asset
- Review and Test in the intended configuration
- Delivery

5.6 Product Developer / Product Architect



- Planning and assembly of SW products from assets and customer specific components
- Analysis of product requirements
- Identification of necessary components (existing and new)
- Requests new requirements from Platform and Customer Specific Developers
- Tracks Customer Specific and Platform Development
- Product Integration
- Product Testing

- Product Release

5.7 Product Manager

- Customer relationship management
- Involved in Product sales and negotiates Product Features with Customer
- Develops project plan for Products
- Controls budget and time tables of product development
- Product Release planning

6 Derivation Process

In this section, we describe the steps to construct a product from the platform. A customer product is never built from scratch. A customer Product is always based on existing assets (requirements, design, code etc.). A customer product will either be based on the platform or some precursor components (from pervious projects within the platform). The product derivation steps described below apply for the case when the Platform is used as the basis of the customer Product.

SYRS:	System Requirements Specification
CRS:	Customer Requirements Specification
SRS:	Software Requirements Specification
RS:	Requirements Specification
P-<rs>:	Platform requirements specification
C-<rs>:	Customer-Specific requirements specification
SYRTP:	System Release Test Plan
SRTP:	Software Release Test Plan
Algo:	A type of subsystem where all firing logic for the sensors is developed. Algo performs a type of software development. Software, Hardware, Sensors, Mechanics

6.1 Rationalising the CRS

The CRS is the primary input for the creation of the C-SYRS. The CRS may consist of an entire collection of requirements specification documents. In general, it will contain both system-level requirements and also requirements particular to the hardware components buried deep in the system design.

The CRS must be translated from a customer document to an internal organisational document, this process is also known as rationalising the requirements (see Figure 3). To translate the customer document the Product Manager must adapt the language within the CRS to the organisational language and change the structure to fit internal organisational documentation structure. To perform requirements translation the Product Manager needs to know both the internal organisational language and the customer language. The Product Manager uses the Product Glossary as a reference during the translation process. The Product Manager does not need detailed technical knowledge of the platform.

The output of this step is the Customer System Requirement Specification (C-SYRS). The C-SYRS contains cross-discipline requirements.

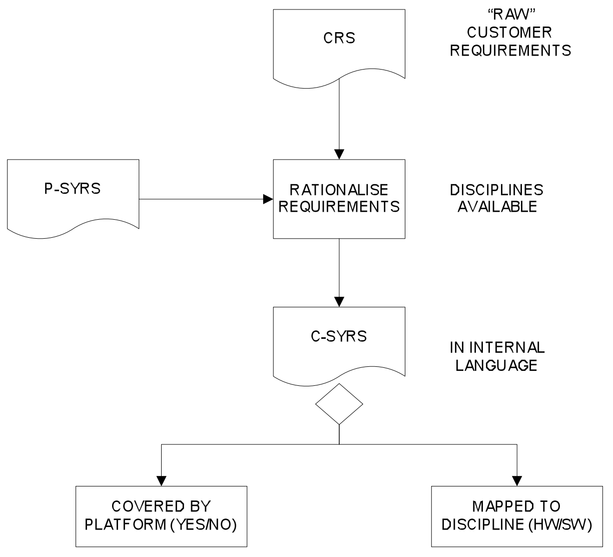

CRS
"RAW" CUSTOMER REQUIREMENTS
P-SYRS
RATIONALISE REQUIREMENTS
DISCIPLINES AVAILABLE
C-SYRS
IN INTERNAL LANGUAGE
COVERED BY PLATFORM (YES/NO)
MAPPED TO DISCIPLINE (HW/SW)

Figure 3 Creating the C-SYRS

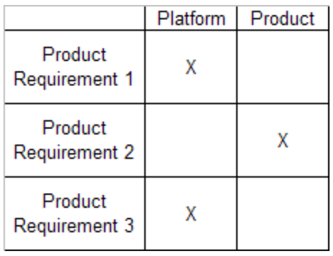
6.2 Coverage Analysis

The Product Manager performs a coverage analysis examination on the C-SYRS (see Figure 4). Coverage Analysis is a comparison of the C-SYRS and the P-SYRS. The Product Manager uses the tool DOORS to assist in requirements traceability. Through

Coverage Analysis the Product Manager discovers which customer requirements are covered by the platform.

Coverage analysis can prove complicated for the Product Manager. The P-SYRS contains requirements at the system level, whereas the C-SYRS may contain requirements from different design levels. Requirements from the C-SYRS and the P-SYRS may have different granularity. As a consequence, it may be difficult to compare such requirements. The C-SYRS may contain requirements, which are not contained in the platform; it may not be clear how such requirements should be handled.

The Product Manager also needs extensive domain experience. If specific requirements cannot be completely satisfied, they are broken into smaller requirements and then mapped to specific components. This additional information is now included in the C-SYRS.

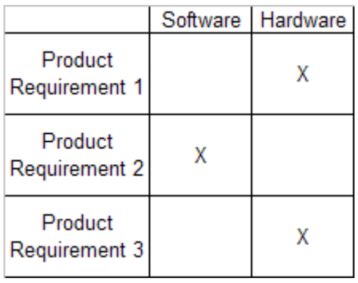


	Platform	Product
Product Requirement 1	X	
Product Requirement 2		X
Product Requirement 3	X	

Figure 4 Contribution of Coverage Analysis to C-SYRS

6.3 Software/Hardware Mapping

The Product Architect defines where C-SYRS requirements can be implemented (see Figure 5). The results of this task are documented in the C-SYRS.



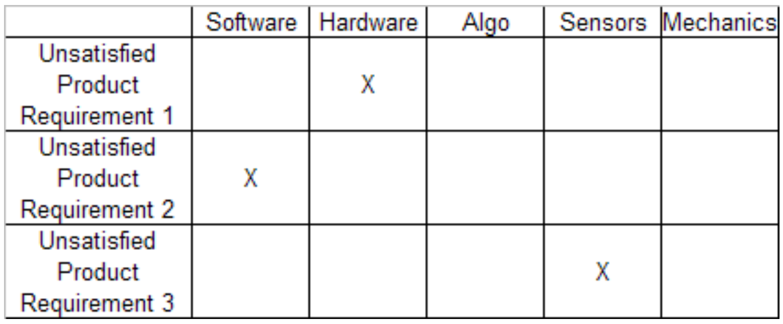
	Software	Hardware
Product Requirement 1		X
Product Requirement 2	X	
Product Requirement 3		X

Figure 5 Contribution of Software/Hardware Mapping to C-SYRS

6.4 Discipline Mapping

The primary goal of discipline mapping is to map unsatisfied requirements (requirements which cannot be implemented through a configuration of platform components) to specific teams (see Figure 6) e.g. Algo, hardware, software, sensors, mechanics.

The Product Manager performs discipline mapping (see Figure 7). Discipline mapping involves allocating requirements from the C-SYRS to different organisational disciplines within the platform, such as hardware, algo (algorithms) or software disciplines. Discipline mapping should be a quick process, the emphasis is on speed not precision. The Product Manager should be able to perform discipline mapping without the involvement of the Product Architect.



	Software	Hardware	Algo	Sensors	Mechanics
Unsatisfied Product Requirement 1		X			
Unsatisfied Product Requirement 2	X				
Unsatisfied Product Requirement 3				X	

Figure 6 Contribution of Discipline Mapping to C-SYRS

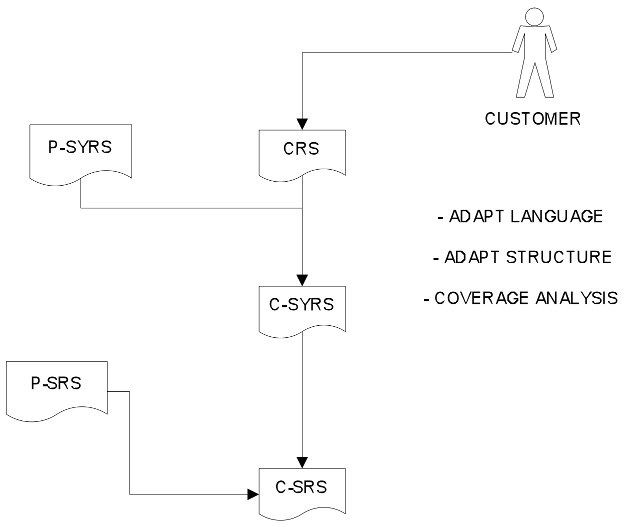


Figure 7 Rationalising the requirements

6.5 Creating the C-SYRTP

The Product Manager creates the Customer System Requirements Test Plan (C-SYRTP) based on the C-SYRS (see Figure 8). The Product Manager writes test cases for the requirements contained in the C-SYRS. As a starting point the P-SYRTP is used. Particular test cases can be added and deleted from the P-SYRTP in a similar way to that used in coverage analysis. This forms the basis for the C-SYRTP. The

Product Manager utilizes the DOORS tool to map each test case to a specific customer requirement.

The output of this step is the C-SYRTP.

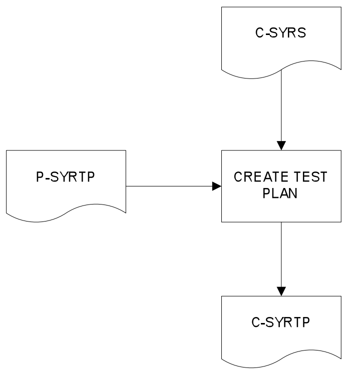


Figure 8 Creating the System Test Plan

6.6 Scope Requirements Implementation

The Product Architect scopes customer requirements deciding if the customer requirement should be implemented in the product or if a request for platform implementation should be made (see Figure 9). Market forecasts, resources, implementation costs and Intellectual property issues all influence the Product Architects scoping decision. Intellectual property becomes an issue when a customer wants a customer-only-solution and does not want their features to become part of the general platform features. The Product Architect uses impact analysis models, decisions models and effort estimation models to assist in the decision making progress.

It should be noted that only requirements not covered by the platform are relevant input here.

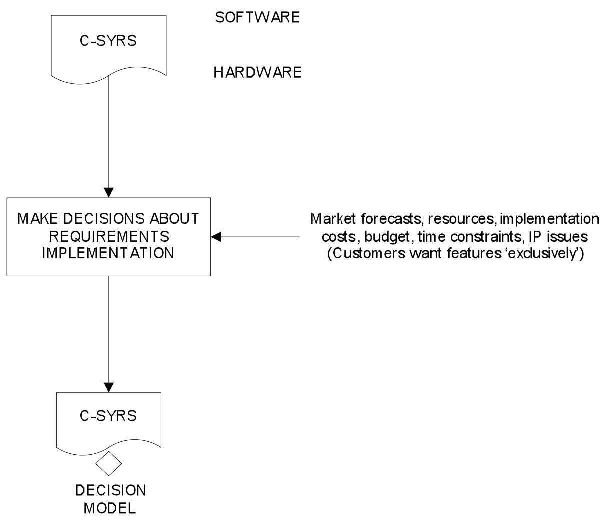

SOFTWARE
C-SYRS
HARDWARE
MAKE DECISIONS ABOUT REQUIREMENTS IMPLEMENTATION
Market forecasts, resources, implementation costs, budget, time constraints, IP issues (Customers want features 'exclusively')
C-SYRS
DECISION MODEL

Figure 9 Deciding requirements implementation

6.7 Exploring the C-SYRS

The Product Manager separates the requirements contained in the C-SYRS into the platform and product specific requirements. The requirements destined for the platform are added to the P-SYRS. As a result of the platform requirements changing the P-SRS and the P-HRS must be changed also to reflect the amended P-SYRS. These changes are based on the platform system architecture (see Figure 10).

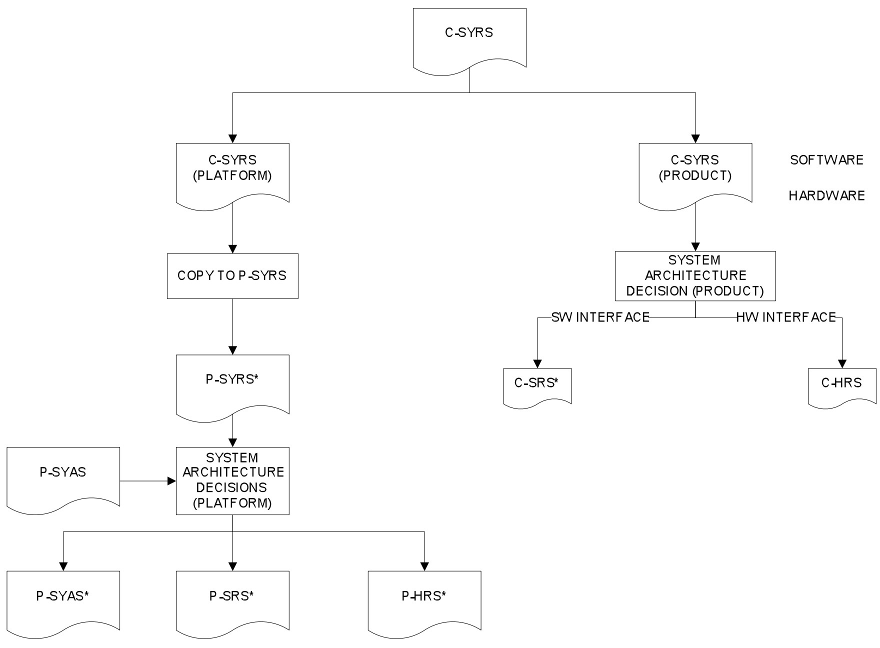


Figure 10 Separating the C-SYRS

Based on the product requirements the Product Architect makes architectural decisions. In particular the C-SRS and C-HRS will need to be updated to reflect among other things interface requirements to allow communication between hardware and software components.

The Product Architect should be able to demonstrate evidence of traceability for System Architecture Decisions including:

- Making decisions relating to product architecture
- Deriving Platform Design requirements for SW + HW (P-SRS/P-HRS)
- Link new requirements to P-SYRS

At this step, both the platform and product teams are implementing certain customer requirements. A type of race condition is sometimes used in this situation. See Figure 11 on configuration and release management within the platform. An example of the race condition can be found in the warning lamp example, where both product and platform race for the single pin in the platform component.

In Figure 11, we can see how configuration and release management operates in the platform. The platform line contains mainline platform artefacts. When the customer requirements are received, a customer specific architecture is derived from the mainline platform architecture. This becomes a separate, customer specific, branch of the platform. Concurrent to this, the product team is developing a product specific development line. The product specific and customer derived platform line are merged when development finishes. The platform development which is deemed to have reuse potential is merged back into the platform mainline.

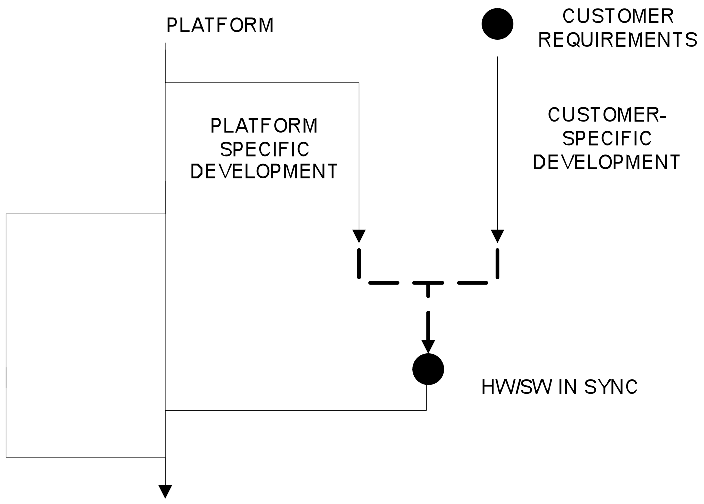


Figure 11 Configuration and release management in SPL

Both the Platform and Product Architect use DOORS[1] for analysis and configuration within the product and platform architecture. AMEOS[2] is used for project management aspects, where traces from / to:

- C-SYRS+
- C-SRS / C-HRS
- P-SYRS
- P-SRS / P-HRS

[1] http://www.telelogic.com/corp/products/doors/
[2] http://www.ameos.com/en.html

6.8 Define and Link Test Cases

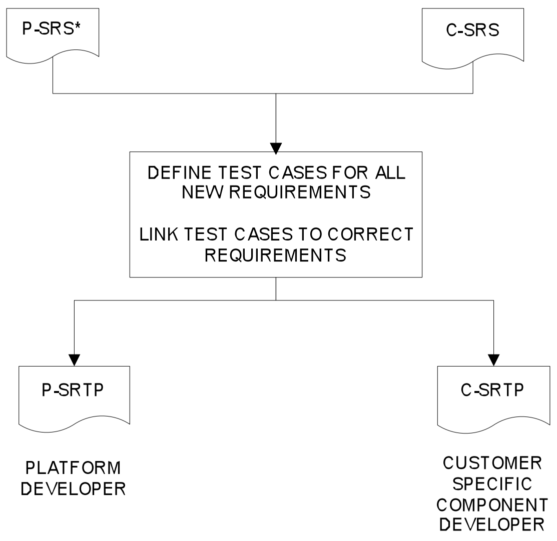

P-SRS*
C-SRS
DEFINE TEST CASES FOR ALL NEW REQUIREMENTS
LINK TEST CASES TO CORRECT REQUIREMENTS
P-SRTP
C-SRTP
PLATFORM DEVELOPER
CUSTOMER SPECIFIC COMPONENT DEVELOPER

Figure 12 Define and Link Test Cases

For the platform customer specific components and the product only components, test cases are defined and linked. The Platform Developer is responsible for writing test cases and linking them to particular platform requirements. The Customer Specific Component Developer is responsible for writing test cases for customer specific components and linking these test cases to specific customer components.

Both the Platform Developer and Customer Specific Component Developer use DOORS to facilitate the linking of test cases to requirements.

6.9 Create Initial Baseline in MKS

The Product Manager takes the C-SYRS and from it, makes amendments to the P-SRS and the C-SRS. The amendments include the addition of platform requirement requests from the Product Managers customers. The product specific requirement requests form the C-SRS (see Figure 13).

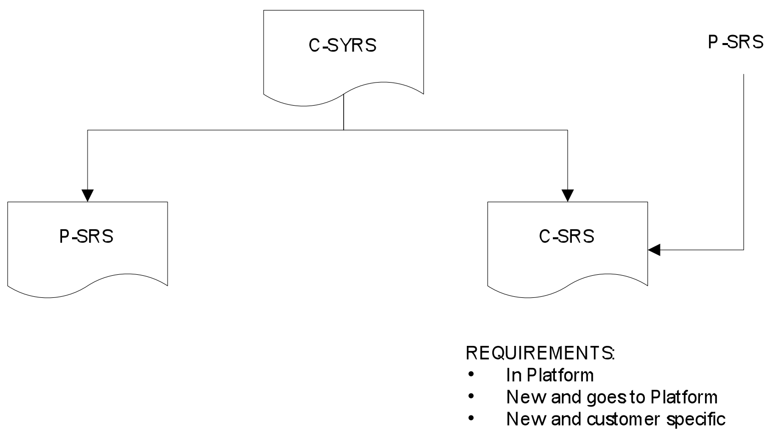


Figure 13 The scoping of requirements in the C-SYRS

Take the example shown in Table 1. Given six requirements in the C-SYRS, each sample requirement is scoped and falls under one of the following headers: The requirements are covered by the platform, the requirements result in new platform requirements or the requirements are specified as product specific.

	Covered by scope of Platform?	New Platform Requirement?	New Product Requirements?
Requirement 1	YES		
Requirement 2		YES	
Requirement 3		YES	
Requirement 4			YES
Requirement 5	YES		
Requirement 6			YES

Table 1 Six example requirements in the C-SYRS

The Product Team use a configuration file to configure the generic platform components into platform product components. The configuration file describes the properties of the product specific version. The new customer specific components are integrated with the configured platform components to form a partial product configuration. Figure 13 graphically illustrates this process.

6.10 Creating the Product Baseline

The Platform Manager amends the P-SRS to include the new requested requirements. The Platform Architect takes the P-SRS and uses it to establish a customer platform baseline in MKS. This involves extending the existing platform software architecture to facilitate new platform requirements. As a result of deriving a product baseline from the Platform Architecture, a modified P-SAS is created to reflect the changes.

To form the initial baseline the Platform Architect copy relevant parts from the platform and precursor customer projects. Unnecessary components are removed. The remaining components are modified as necessary in order to get a compatible system.

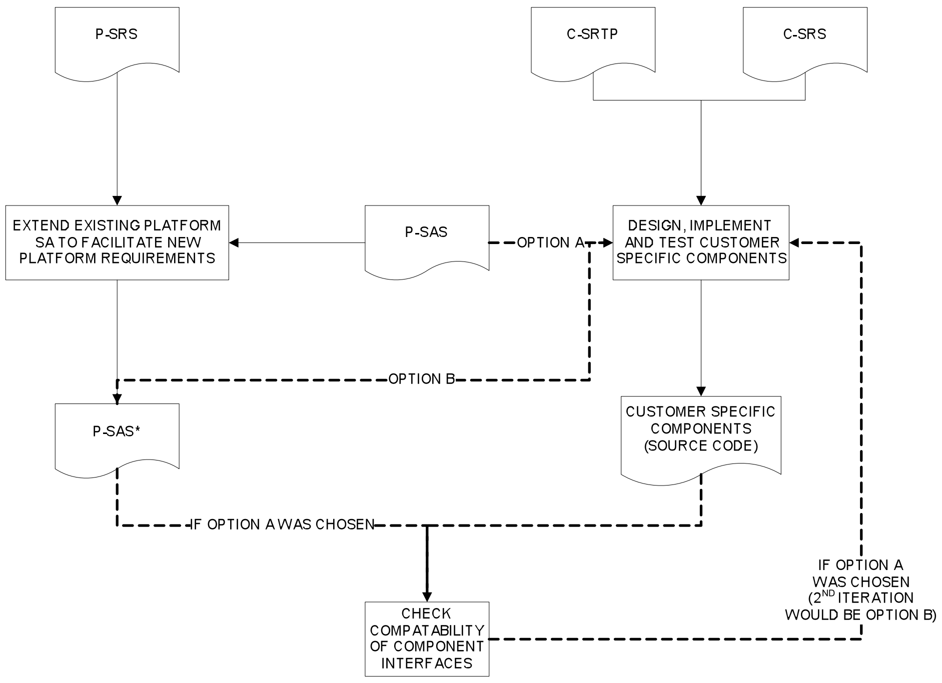

P-SRS
C-SRTP
C-SRS
EXTEND EXISTING PLATFORM SA TO FACILITATE NEW PLATFORM REQUIREMENTS
P-SAS
OPTION A
DESIGN, IMPLEMENT AND TEST CUSTOMER SPECIFIC COMPONENTS
OPTION B
P-SAS*
CUSTOMER SPECIFIC COMPONENTS (SOURCE CODE)
IF OPTION A WAS CHOSEN
IF OPTION A WAS CHOSEN (2ND ITERATION WOULD BE OPTION B)
CHECK COMPATABILITY OF COMPONENT INTERFACES

Figure 14 Creating initial product baseline

Concurrently to the Platform Team deriving a customer product from platform architecture, the Product Team are designing, implementing and testing customer specific components.

6.11 Design and Implementation of new Platform Components

The Platform Architect uses the P-SRS and the P-SAS to design and allocate platform component development. The Platform Software Design Specification (P-SDS) is used to document the planned component implementation as seen by the Platform Architect. It is the responsibility of the Platform Developer to implement the requested components.

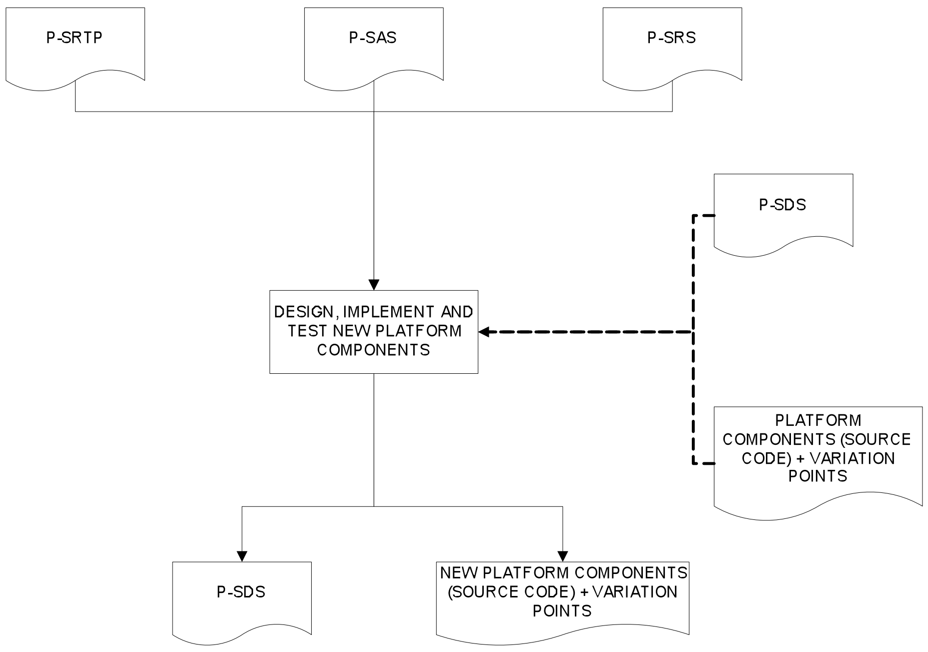

P-SRTP
P-SAS
P-SRS
P-SDS
DESIGN, IMPLEMENT AND TEST NEW PLATFORM COMPONENTS
PLATFORM COMPONENTS (SOURCE CODE) + VARIATION POINTS
P-SDS
NEW PLATFORM COMPONENTS (SOURCE CODE) + VARIATION POINTS

Figure 15 Design and Implementation of new platform components

The Platform Developer creates the new platform components with variation points as stated in P-SDS. The Platform Architect will use configuration management tools to facilitate in component design. Until the Platform Team has implemented the requested product platform amendments, the Product Team cannot complete product construction.

6.12 Tailoring the Platform Architecture

The Product Architect makes a copy of the latest platform version. This new version contains the platform changes implemented in Design and Implementation of New Platform Components. The Product Architect tailors the platform architecture to form the product architecture and resolves variation points according to customer requirements.

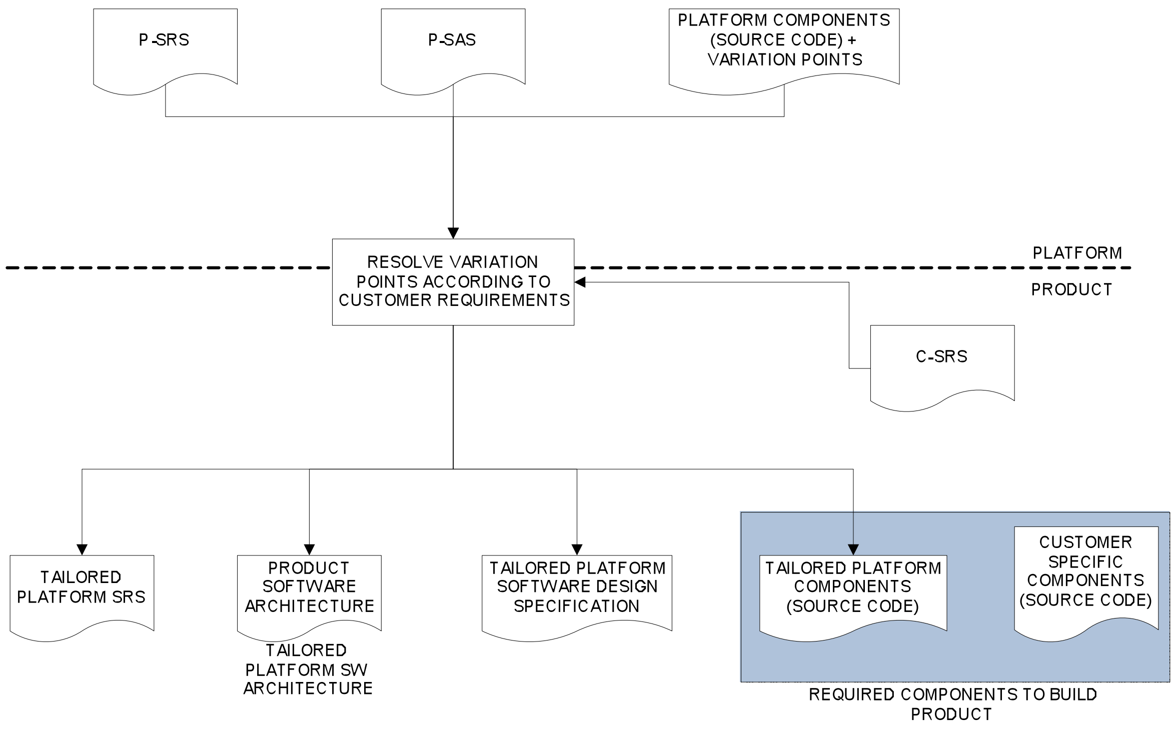

P-SRS
P-SAS
PLATFORM COMPONENTS (SOURCE CODE) + VARIATION POINTS
RESOLVE VARIATION POINTS ACCORDING TO CUSTOMER REQUIREMENTS
PLATFORM
PRODUCT
C-SRS
TAILORED PLATFORM SRS
PRODUCT SOFTWARE ARCHITECTURE
TAILORED PLATFORM SW ARCHITECTURE
TAILORED PLATFORM SOFTWARE DESIGN SPECIFICATION
TAILORED PLATFORM COMPONENTS (SOURCE CODE)
CUSTOMER SPECIFIC COMPONENTS (SOURCE CODE)
REQUIRED COMPONENTS TO BUILD PRODUCT

Figure 16 Tailoring the Platform Architecture

6.13 Integrating the Product

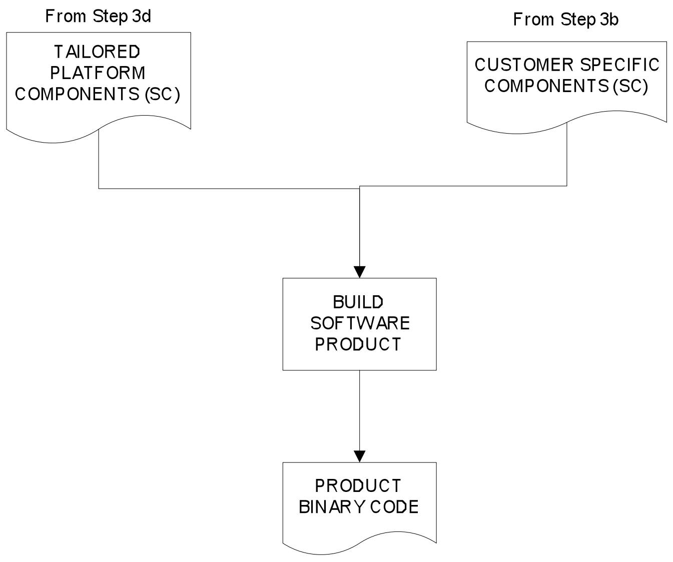

From Step 3d
TAILORED PLATFORM COMPONENTS (SC)
From Step 3b
CUSTOMER SPECIFIC COMPONENTS (SC)
BUILD SOFTWARE PRODUCT
PRODUCT BINARY CODE

Figure 17 Product Integration

The Product Developer integrates the product taking both the tailored platform components and customer specific components. The Product Developer builds the software product and the resulting output is product binary code. Typically the compiler, which the Product Developer uses to build the product, is optimised to specific hardware architecture.

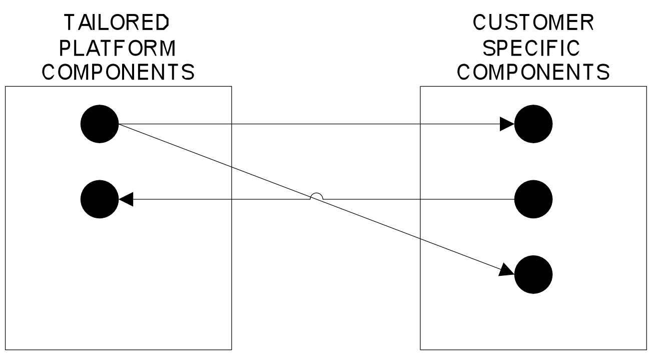


Figure 18 Linkage of platform and customer specific components

6.14 Integration Testing

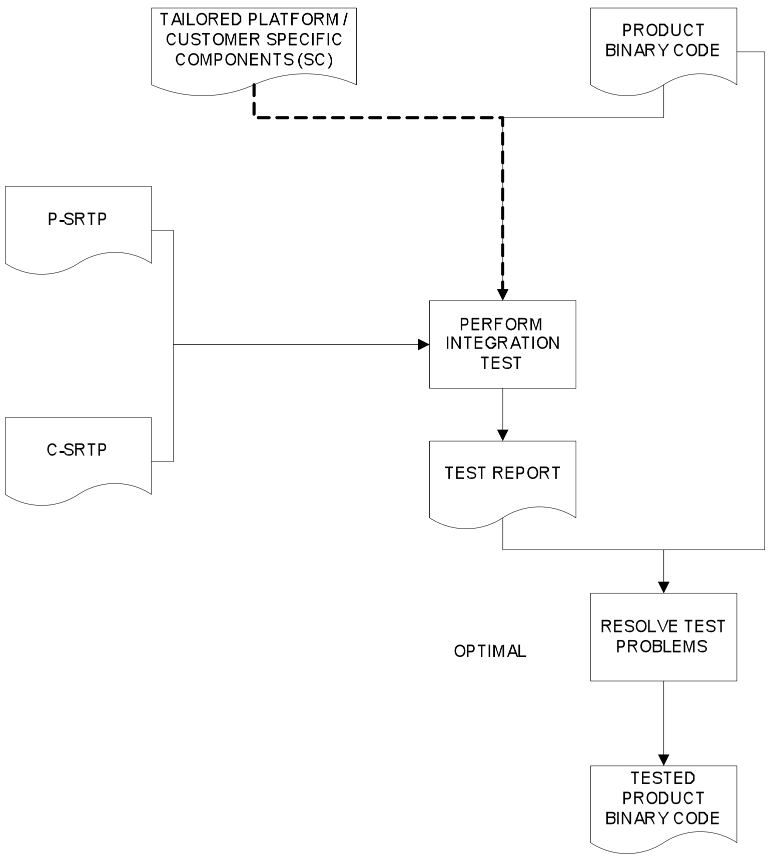


Figure 19 Performing Integration Testing

The Product Developer performs integration testing on the product using both platform and product test plans. The P-SRTP and C-SRTP contain both unit test cases and integration test cases.

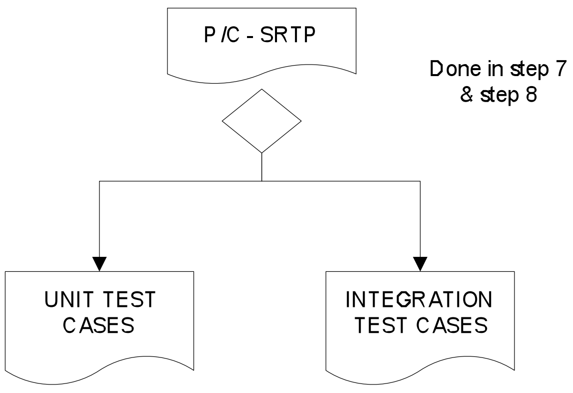


Figure 20 the contents of P-SRTP and C-SRTP

The result of the integration testing is a testing report with all diagnosed test problems. The Product Developer uses a workbench to test the software in its target hardware environment.

7 Discussion

The main observations obtained from the case study on industrial product derivation practices were:

- Additional Development Disciplines
- Additional Roles and Tasks
- Platform-Product Synchronisation
- Use of Documentation

7.1 Additional Development Disciplines

The organisational structure for a particular business unit engaged in product derivation is broken into three broad disciplines; software, hardware and mechanics. Within each of these disciplines there are further sub-disciplines. For instance, the hardware discipline has a microcontroller team and an ECU (Electronic Control Unit) team. The mechanics discipline has housing, mechanical quality and interfaces and plugs teams. The software discipline had basic software and algorithms teams.

7.2 Additional Roles and Tasks

During the case study a number of product derivation roles were observed. For example, the software product team consists of architects, developers, integrators, testers, and customer specific component developers. These roles are replicated across the independent product sub-discipline teams and platform teams. Moreover, similar roles exist for hardware and mechanics.

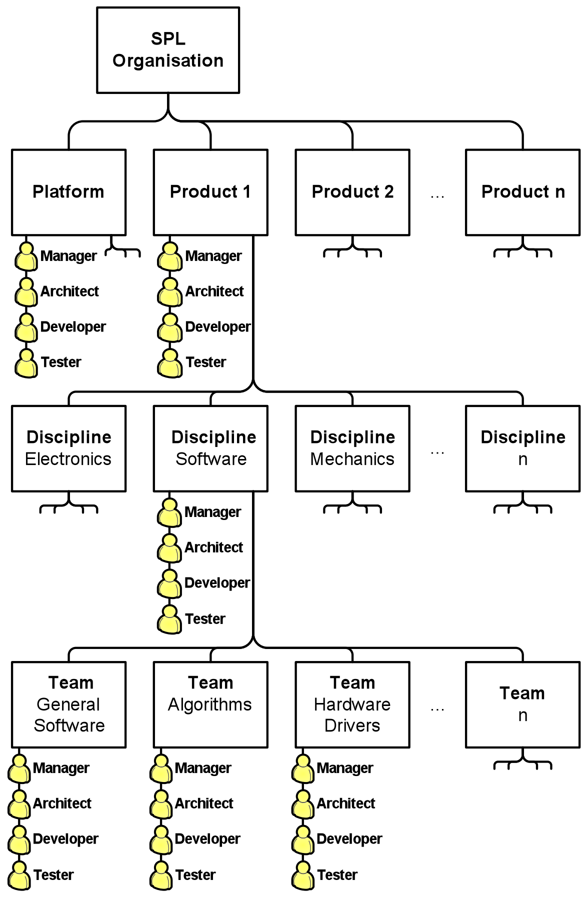


Figure 21 Role Structures

These intricate role structures are reflected by appropriate communication and task structures. For instance, the allocation of requirements to responsible teams has to consider the various disciplines and sub-disciplines. This requires a higher degree of granularity for requirements management tasks than originally envisaged. This can be observed when the case study company starts a product-specific project. During the early phases, the customer requirements are translated into a set of internal company documents. These documents are processed and augmented during various tasks where requirements are analysed for reuse potential and then assigned to responsible disciplines and sub disciplines.

Another consequence of this distributed development across both disciplines and platform and product teams is the raised importance of modularisation. Consequently, interface management is performed as an explicit task and encapsulation is a key design property for component development; a software component should ideally be independent of how a sensor, actuator or microcontroller works internally.

7.3 Platform-Product Synchronisation

Within the case study company, product development requires a high degree of coordination and communication as the heavy dependencies across disciplines and the platform product divide is managed. In the observed platform product dependencies are illustrated.

The product team both designs and implements customer specific components based on the customer requirements. The platform team receives the platform software requirements containing the required extensions to the existing platform in order to facilitate the new customer requirements. Both the customer-specific and platform development is occurring in parallel. The product team needs to interface correctly with the new platform release. Here, the product team can choose between two alternative development strategies. Option 1 is to design and implement customer specific components using the *current* platform release, which has not yet been updated, as a basis for development. Consequently, when the new platform architecture is released the product team has to check the compatibility of the developed components with the new architecture.

Option 2 for the product team is to wait for the updated platform release. This is suggested when potentially large compatibility issues are expected with the risk of wasted development effort.

A third hybrid option, not illustrated in Figure 22, is for the product team to first negotiate a platform interface with the platform team before proceeding to develop in parallel against the platform team. Alternatively, the product team can make assumptions on expected interface changes, and work from these expectations. After the updated platform release the product team check the compatibility of the developed components with the new architecture.

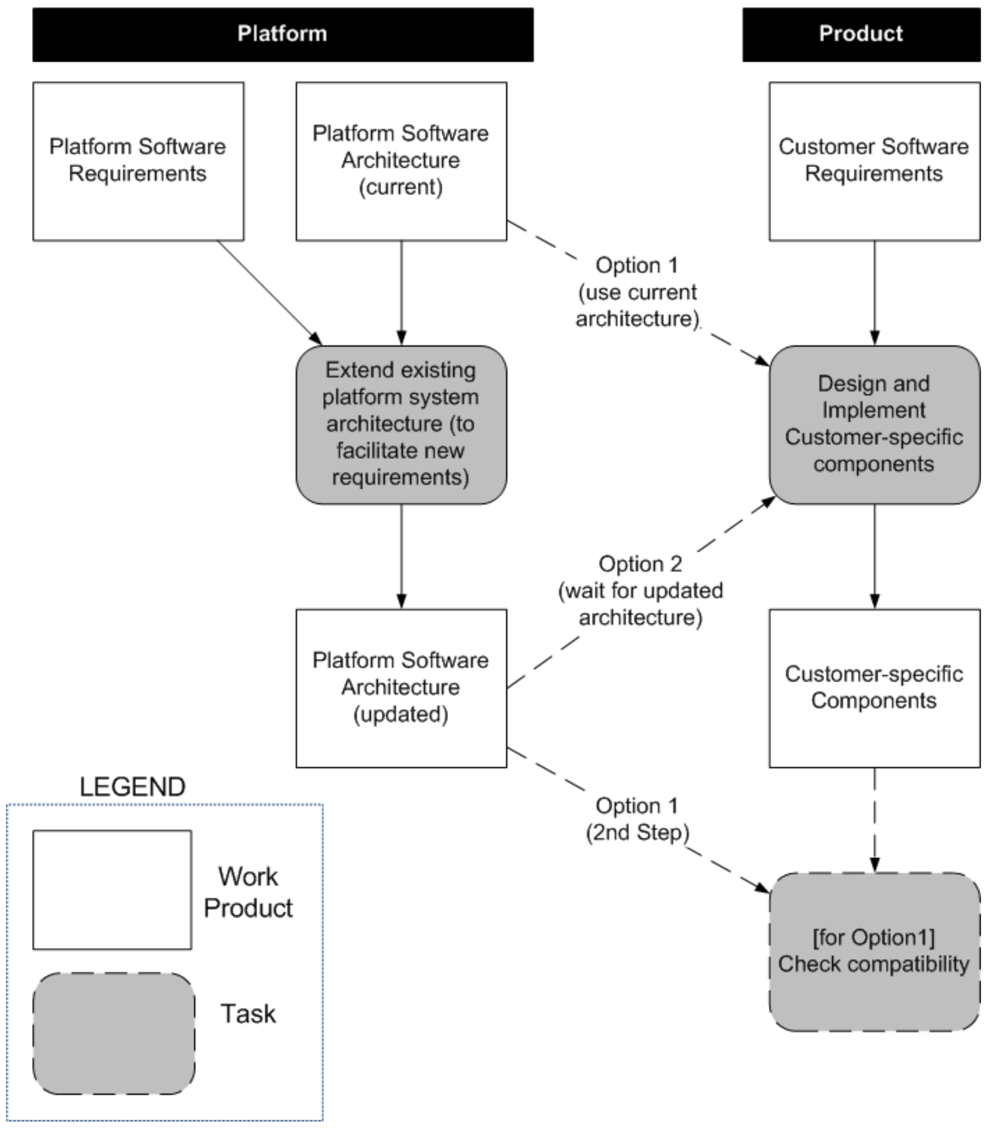

Platform
Product
Platform Software Requirements
Platform Software Architecture (current)
Customer Software Requirements
Option 1 (use current architecture)
Extend existing platform system architecture (to facilitate new requirements)
Design and Implement Customer-specific components
Option 2 (wait for updated architecture)
Platform Software Architecture (updated)
Customer-specific Components
LEGEND
Work Product
Task
Option 1 (2nd Step)
[for Option1] Check compatibility

Figure 22 Synchronisation Caused by Platform-Product Dependency

These outside dependencies that the product team must handle are a reoccurring process pattern within product derivation. Similar dependencies can be seen during software integration with the hardware modules. The software product team can choose one of the development strategies similar to those described, for handling the hardware interfaces during parallel software and hardware development.

7.4 Use of Documentation

The case study company relies heavily on documentation to drive the product derivation process. Documentation is used to facilitate communication and synchronise development between the product and platform teams, between the different hardware, software and mechanical disciplines and also between the sub-disciplines. It is also used as a milestone to plot project progress and as a driver to trigger certain tasks within the project. Additionally, in certain domains, law requires evidence of due process. Some documents satisfy this condition.

8 Conclusion

The observations from this technical report are based on data collected on the product derivation process of a software-intensive automotive product lines. The observations from this study are detailed here. The observations are focussed on the organisational overview of the SPL, roles and responsibilities and the product derivation process within the company.

During the case study a number of product derivation roles were observed. For example, the software product team consists of architects, developers, integrators, testers, and customer specific component developers. Product development requires a high degree of coordination and communication as the heavy dependencies across disciplines and the platform product divide is managed. The case study company relies heavily on documentation to drive the product derivation process. The findings from this case study were used in the development of Pro-PD [13, 14].

The report contributes to an improved understanding of industrial product derivation practice. It can be considered a source of empirical evidence for derivation practice and thus can enrich existing theory on product derivation practice.

9 References

1. Clements, P. and L. Northrop, *Software Product Lines: Practices and Patterns*. 2001, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
2. Hotz, L., A. Gunter, and T. Krebs, *A Knowledge-based Product Derivation Process and some Ideas how to Integrate Product Development*, in *Proc. of Software Variability Management Workshop*. 2003: Groningen, The Netherlands.
3. Deelstra, S., M. Sinnema, and J. Bosch, *Product Derivation in Software Product Families: A Case Study.* Journal of Systems and Software, 2005. **74**(2): p. 173-194.

4. Deelstra, S., M. Sinnema, and J. Bosch, *Experiences in Software Product Families: Problems and Issues During Product Derivation*, in *Software Product Lines, Third International Conference*. 2004, Springer: Boston, MA, USA.
5. Griss, M.L., *Implementing Product-Line Features with Component Reuse*. ICSR-6: Proceedings of the 6th International Conference on Software Reuse. 2000, London, UK: Springer-Verlag. 137--152.
6. Rabiser, R., P. Grünbacher, and D. Dhungana, *Supporting Product Derivation by Adapting and Augmenting Variability Models*, in *11th International Software Product Line Conference*. 2007: Kyoto, Japan.
7. Walsham, G., *Interpreting Information Systems in Organizations*. 1993, New York, NY, USA: John Wiley & Sons, Inc. 286.
8. Orlikowski, W. and J. Baroudi, *Studying Information Technology in Organizations: Research Approaches and Assumptions.* Information Systems Research, 1991.
9. Perry, D., S.E. Sim, and S. Easterbrook, *Case Studies for Software Engineers*, in *Proceedings of the 26th International Conference on Software Engineering*. 2004.
10. Yin, R., *Case Study Research : Design and Methods*. 2003, Beverly Hills, CA: SAGE Publications.
11. Hammersley, M., R. Gomm, and P. Foster, *Case Study Method: Key Issues, Key Texts*. 2000, London: Sage Publications.
12. *The SPLC Product Line Hall of Fame*. 3/02/2009]; Available from: http://www.splc.net/fame.html.
13. O'Leary, P., R. Rabiser, I. Richardson, and S. Thiel. *Important Issues and Key Activities in Product Derivation: Experiences from Two Independent Research Projects*. in *Software Product Line Conference*. 2009. San Francisco, CA, USA: Proc. of the 13th International Software Product Line Conference (SPLC 2009).
14. O'Leary, P., F. McCaffery, S. Thiel, and I. Richardson, *An Agile Process Model for Product Derivation in Software Product Line Engineering (in press).* Journal of Software Maintenance and Evolution, 2010.