UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI **INGEGNERIA**

Corso di Laurea Magistrale in **Ingegneria Gestionale**

GESTIONE DEGLI OBIETTIVI NELLO SVILUPPO DI PIATTAFORME DI PRODOTTO INNOVATIVE IN UN CONTESTO AGILE

SCOPE MANAGEMENT OF NEW PRODUCT PLATFORM, IN A CONTEXT OF AGILE DEVELOPMENT

Relatore: Chiar.mo

Prof. **Filippo Emanuele CIARAPICA**

Tesi di Laurea di:

Luca CAFFARINI

Correlatore: Chiar.mo

Prof. **Rui ABRANTES**

A.A. **2019 / 2020**

Table of Contents

Abstract

Nel tempo il mercato ha subito una profonda trasformazione e le aziende sono chiamate a rispondere alle sempre più elevate esigenze dei clienti e ad un sempre maggiore grado di personalizzazione del prodotto. L'uso delle piattaforme di prodotto è stato riconosciuto come un mezzo strategico per ottenere la personalizzazione di massa.

In generale si possono identificare due tipi di processi di sviluppo di nuovi prodotti: Tradizionale e Agile. Quando si considera un processo di sviluppo tradizionale ci si riferisce a modelli come Waterfall, dove la modalità di lavoro è fortemente orientata all'esecuzione di passaggi sequenziali. Al contrario dei metodi tradizionali, abbiamo processi di sviluppo Agile, come Extreme Programming (XP) o SCRUM. Questi ultimi modelli sono composti da una serie di fasi iterative, in cui viene considerata ogni volta una piccola e definita parte del progetto. Nel modello Waterfall, si presume che tutti i requisiti del prodotto siano identificati nella fase iniziale, prima delle fasi di progettazione e implementazione, ed è difficile modificare il prodotto nelle fasi finali. L'approccio Agile consente di sviluppare prodotti in modo incrementale e iterativo insieme al feedback dei clienti. In questo modo è possibile reagire agli imprevisti e il prodotto può subire modifiche anche nelle fasi finali o quando è già ultimato.

La presente tesi mira ad analizzare l'utilizzo degli approcci Agile nei processi di sviluppo della Product Platform. Questo studio è stato condotto attraverso la ricerca, nella letteratura scientifica, di casi reali che potrebbero mostrare l'uso di metodologie agili nello sviluppo di piattaforme di prodotto nel dominio dei prodotti fisici e software.

Il Capitolo 1 spiega i metodi di sviluppo del prodotto tradizionali e agili, mostrandone le differenze.

Il Capitolo 2 introduce una delle metodologie di sviluppo agile più popolari, chiamata Scrum.

Il Capitolo 3 analizza principalmente le metodologie utilizzate nello sviluppo di piattaforme di prodotto fisico, il tipo di informazioni gestite per il suo sviluppo e miglioramento e i casi studio.

Il Capitolo 4 presenta la Software Product Line (SPL), una teoria di sviluppo di piattaforme software, e le principali metodologie applicate.

Il Capitolo 5 indaga l'uso di metodologie agili all'interno della linea di prodotti software analizzando le metodologie proposte e i casi studio.

Il Capitolo 6 analizza l'uso di metodologie agili nello sviluppo di prodotti fisici e l'uso di framework agili su larga scala.

Nel capitolo Discussione vengono mostrati i principali risultati dello studio.

Infine, il capitolo 7 rivela le conclusioni.

Questa ricerca mira a trovare una connessione tra lo sviluppo della piattaforma del prodotto e lo sviluppo agile nel campo dei prodotti fisici e software. In particolare, lo scopo dello studio è stato quello di trovare casi di studio dell'applicazione dei framework Agili, come Scrum, nella gestione degli obiettivi della piattaforma di prodotto. Come risultato viene dimostrata, grazie ai casi di studio analizzati, la possibilità di utilizzare le metodologie agili per lo sviluppo delle piattaforme software. È stato mostrato come la metodologia agile combinata con un'architettura della linea di prodotto flessibile possa fornire una consegna tempestiva e continua del software. Inoltre, è stata dimostrata la possibilità di creare piattaforme software pronte ad accogliere i cambiamenti dei requisiti in qualsiasi momento, anche nelle fasi finali dello sviluppo.
La ricerca ha rilevato una mancanza di letteratura sull'applicazione dei framework Agile nello sviluppo di piattaforme di prodotti fisici e in generale vi è una carenza di studi empirici che affrontano l'impiego di framework Agili su larga scala.
Inoltre, sono state trovate somiglianze tra le metodologie di sviluppo tradizionali di piattaforme per prodotti fisici e le metodologie per lo sviluppo agile di piattaforme software. Date queste somiglianze tra le metodologie analizzate, come suggerimenti per il futuro, il framework Scrum potrebbe essere applicato allo sviluppo di piattaforme di prodotti fisici, come è stato fatto per le famiglie di prodotti software.

Introduction

Over time, the market has undergone a major transformation and companies are called upon to respond to ever higher customer needs and an ever greater degree of product customization.
Being able to cope with short lead times and react to changing requirements is a key factor to create unique products. The use of product platforms has been recognized as a strategic enabler for mass customization.
In general two types of new product development processes can be identified: Traditional and Agile. When we consider a traditional development processes we refer to models such as Waterfall, where the working mode is strongly oriented to the execution of sequential steps. As opposed to the traditional methods we have Agile development processes, such as Extreme Programming (XP) or SCRUM. The latter models are composed of a series of iterative phases, where one small and defined part of the project is considered each time. In the Waterfall model, it is assumed that all the product's requirements are identified in the early stage, before the design and implementation phases, and it is difficult to modify the product in the final stages. The Agile approach permits to develop products

incrementally and iteratively coupled with customer feedback. In this way it is possible to react to the unexpected and the product can undergo changes even in the final phases or when it's already completed.
The present thesis aimed to analyze the use of Agile approaches in Product Platform development processes. This study was carried out through the research, in the scientific literature, of real cases that could show the use of agile methodologies in the development of product platforms in the domain of physical and software products.
The first chapter explain the traditional and agile product development methods.
The second chapter introduce one of the most popular agile development methodologies, called Scrum.
The third chapter mainly analyzes the methodologies used in the development of physical product platforms and the type of information managed for its development and improvement.
The fourth chapter presents the Software Product Line (SPL), a theory of platform-based Software development and some of methodologies applied.
The chapter 5 investigates the use of agile methodologies within the software product line by analyzing proposed methodologies and case studies.
Chapter 6 analyzes the use of agile methodologies in the development of physical products and the use of large-scaled agile frameworks.
In the Discussion chapter the major findings of the study are shown.
Finally, Chapter 7 reveals the conclusions.

1 PRODUCT DEVELOPMENT

1.1 TRADITIONAL METHODS

1.1.1 Waterfall and Stage-gate

The Waterfall Model, also called Classic Life Cycle, is a sequential and linear approach to software development, born in the 1950s, when the software development business began to establish itself. At that time, since no software development methodology was present, the developers were inspired by manufacturing production processes and the construction industries, to obtain a methodology that could be applied to code development in an orderly and less chaotic way.

The first traces of the waterfall model can be found in a 1956 publication by Herbert D. Benington (1983), where he describes a sequential structure formed by different phases, which was used for the development of the complex Semi-Automatic Ground Environment (SAGE) for the American defense. The waterfall method was formally described later by Winston Royce (1970). Although Royce never mentions the word "waterfall" in the article, this methodology is known all over the world with this name due to the particular structure of the various activities that compose it, where the flow of development flows linearly from a phase to the next one. This means that the output produced by the first stage will be the input for what follows. The linear execution of all phases produces the final software product in output. A characteristic of this method is that it is based on the big design up front, for this reason product modifications in the final stages are costly and it is advisable to modify the product in the early stages.

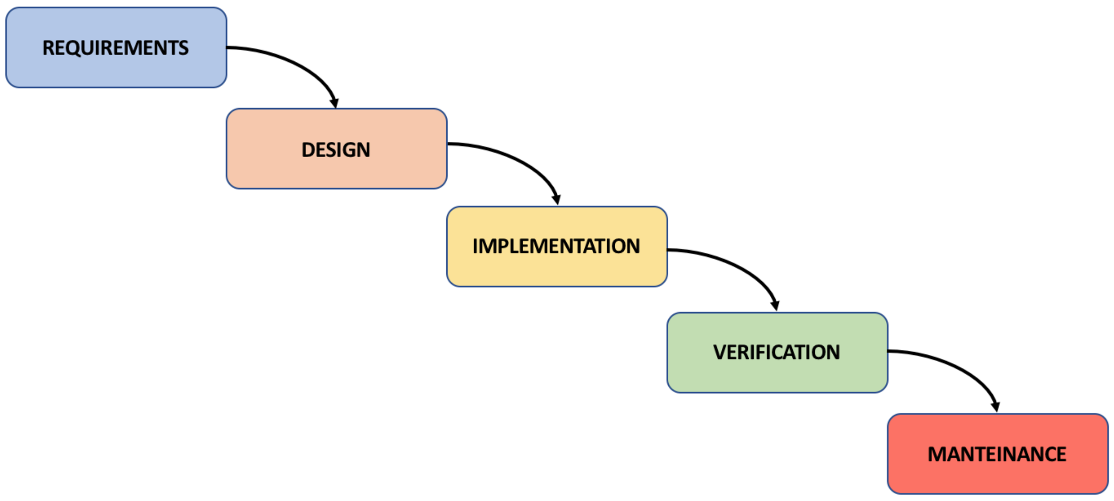


Figure 1 - Waterfall Model

The stage-gate methodology is also a Waterfall methodology, widely used in the development of physical products. This model was designed by Cooper (1990) in the early 1990. It is made up of Stages (or phases) and at the end of each of them it is decided whether or not to continue with the development of that product (Go / Kill decision). At the end of each Stage, the related Gate must review the work and decide if the project can move on to the next Stage. If the passage to the next Stage is not accepted, the project remains in that Stage until the problem is solved.
According to Cooper (2008) the game is won or loss in the front end, i.e. ideation, scoping the project, defining the product, and building the business case. This is the part that most influences the success of the project.

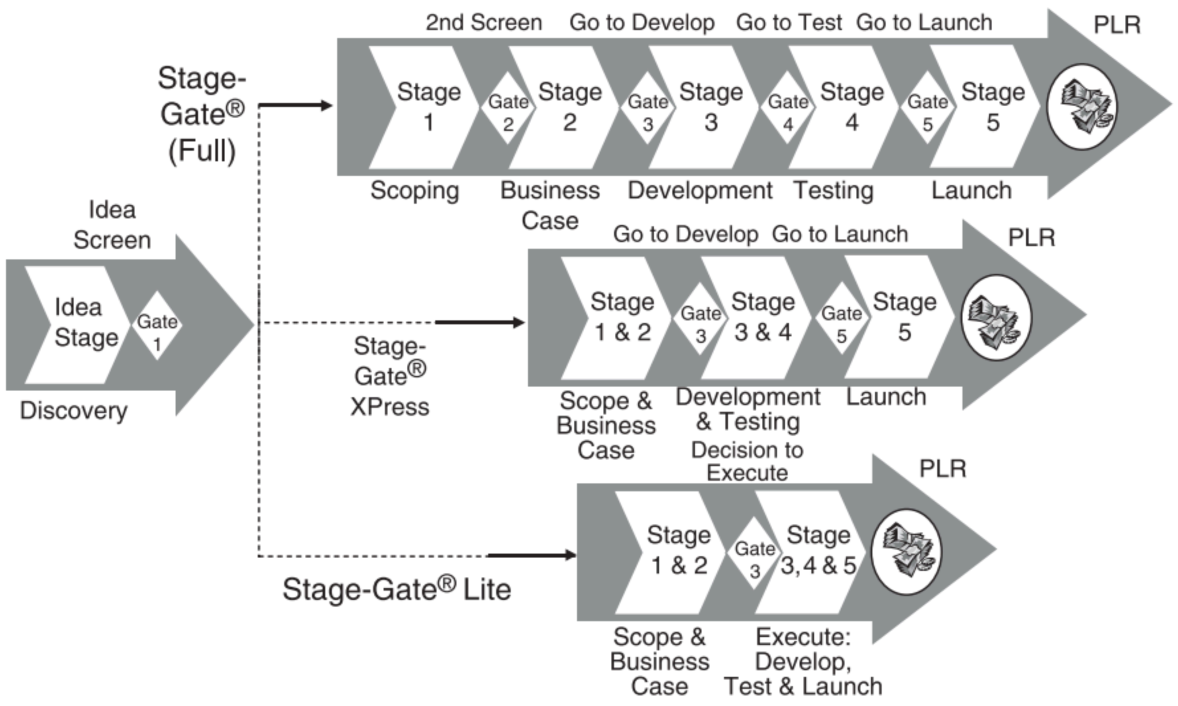


Figure 2 - Stage-gate model versions (Cooper, 2008)

1.1.2 Set-based Concurrent Engineering

Traditional design practice tends to quickly converge on a solution and then modify that solution until the required objectives are met. There is the risk of creating a sub-optimal solution if you start from the wrong place, without considering the time lost to refine that solution. Set-based Concurrent Engineering (SBCE), instead, considers group of possible solutions and shrinks them to obtain a final solution, as shown in Figure 3, drawn by Raudberget (2010). This method may take longer initially to identify the various solutions, but allows to reach the final solution more quickly (Sobek et al., 1999). The SBCE is based on three basic concepts: *Mapping the design space, Integrate by intersection, Establish feasibility before commitment.*

One of the most important idea of Concurrent Engineering (CE) is to carry out activities in parallel, trying to bring more feedback upstream through face-to-face meetings with the purpose of decrease product development and delivery time (Prasad, 1999). According to Levandowski et al. (2014), SBCE can be used to define variants in the product platform design process. Also Johannesson et al. (2017) used this method for the construction of a product platform.

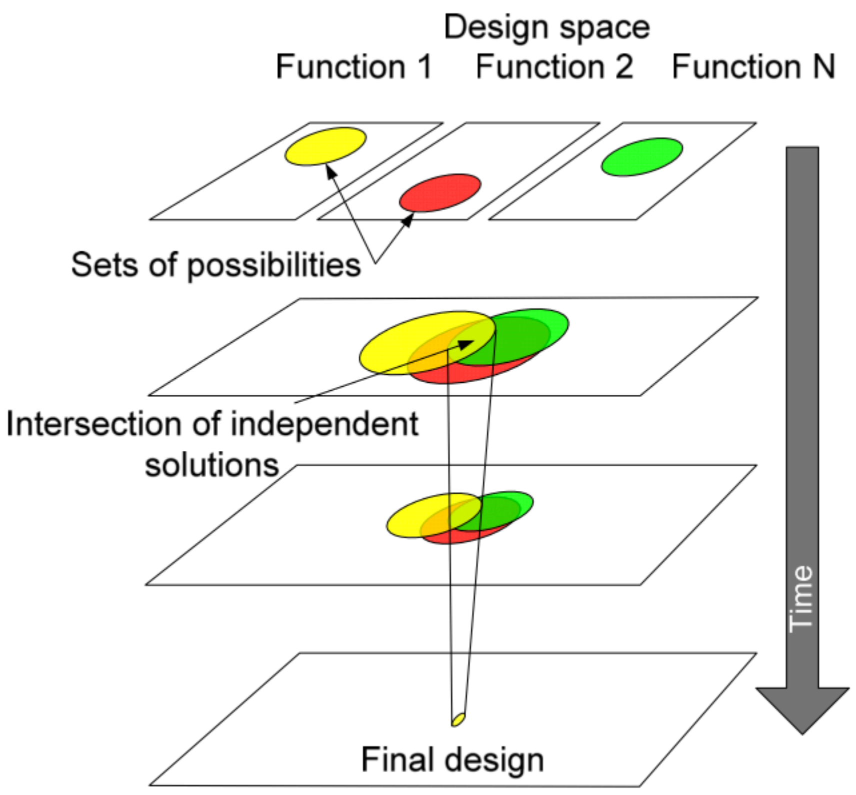


Figure 3 - Principles of SBCE (Raudberget, 2010)

1.2 AGILE METHODS

Agile methodology appeared in 2001 with the conception of the Manifesto for Agile Software Development (Beck et al., 2001). The Agile Manifesto consists of 12 principles:

1. *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."*
2. *"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."*
3. *"Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. "*
4. *"Business people and developers must work together daily throughout the project. "*
5. *"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."*
6. *"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."*
7. *"Working software is the primary measure of progress."*

8. *“Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.“*
9. *“Continuous attention to technical excellence and good design enhances agility.”*
10. *“Simplicity--the art of maximizing the amount of work not done--is essential.”*
11. *“The best architectures, requirements, and designs emerge from self-organizing teams.”*
12. *“At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”*

In most situations, customers are not clear about what they want and for this reason they must be followed with particular attention to try to make them understand the requirements of a product that is able to meet their needs. To achieve this, a strong collaborative bond must be built with customers. One of the main differences with the traditional methodology is that the agile methodology allows the product requirements to be modified during the development phase, they are not fixed, as shown in Figure 4. Furthermore, the agile methodology is incremental and iterative, then new versions of the product are released at frequent intervals.

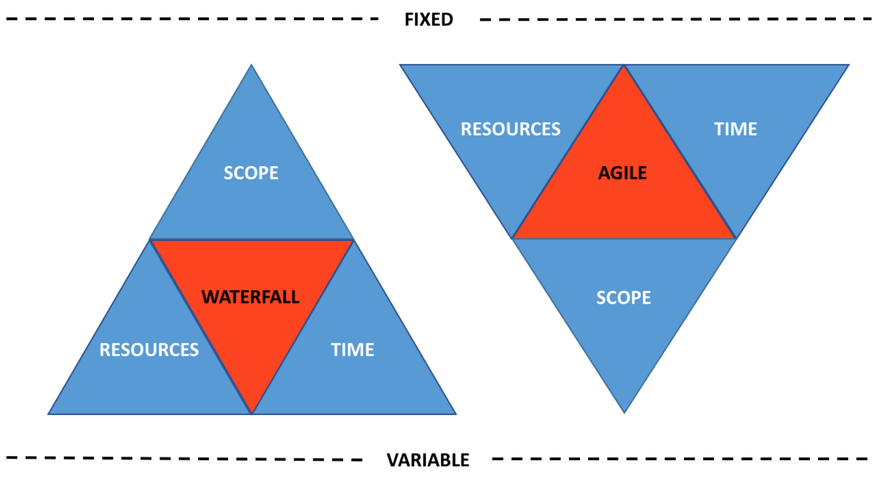


Figure 4 - WATERFALL vs. AGILE

Various agile methodologies have been devised, one of the first and most used is Scrum, explained in Chapter 2. Others methodologies are Extreme programming (XP) and Feature-Driven Development (FDD). Frameworks that allow to scale Agile and Scrum development at large scale exist. The best known are Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS) or Disciplined Agile Delivery (DAD) and Scrum of Scrums (SoS).

2 SCRUM AGILE FRAMEWORK

Scrum is one of the most important and widely used Agile frameworks for software development, conceived by Ken Schwaber and Jeff Sutherland in early 1990s (2020). This method appeared to respond to an ever increasing market competitiveness following the principles of the Agile Manifesto written by Beck et al. (2001). For this reason the Scrum method has as its main objectives the customer satisfaction, the ability to adapt to changes in requirements and the willingness to deliver working products with a certain cadence. This framework applies an iterative, incremental approach combined with Sprint events. It starts normally with a vision of the system to be developed (Schwaber, 2004)

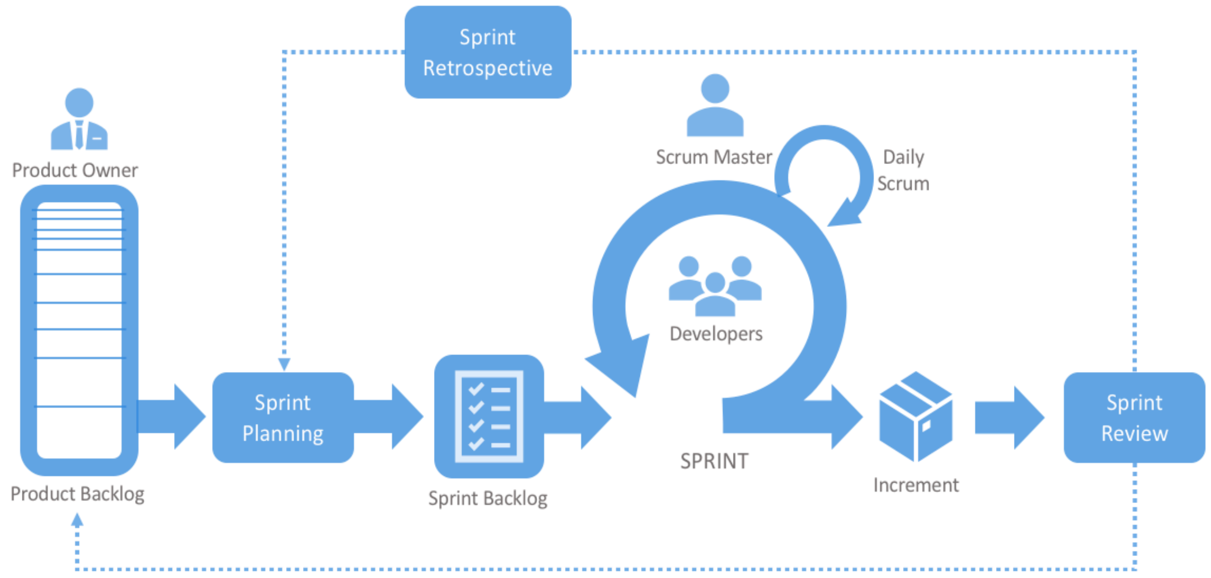


Figure 5 - Scrum Framework overview

2.1 Scrum Team

The Scrum Team is accountable to give life to the product, and since they have to work together on ongoing projects, it is advisable that they develop a strong and close relationship, as peers (Pichler, 2010).

This team is composed of Product Owner, Developers and Scrum Master and it is cross-functional, self-managing and oriented to the Product Goal.

When the team is small, their members communicate and work more efficiently. For this reason is advisable that the team consist of maximum 10 people, and if the number exceeds it is better to split the team into multiple teams (Schwaber & Sutherland, 2020).

2.1.1 Product Owner

The Product Owner is the person that is in charge of building the product vision, managing the product backlog, staying in contact with the stakeholders and cooperating with the team (Pichler, 2010). Schwaber & Sutherland (2020) defines the Product Owner as the person responsible for maximizing the product value.

Furthermore he is the only representative for the stakeholders, he interprets customer needs and translates them into product backlog items.

He guarantees that:

- The elements of the Product Backlog are clearly expressed;
- Product Backlog items are ordered to better achieve objectives and missions;
- The value of the work done by the Team is optimized;
- The Product Backlog is visible, transparent and clear to all and shows what the Scrum Team will work on next;
- Items in the Product Backlog are understood to the required level by the Development Team.

2.1.2 Developers

Developers are an important part of the Scrum Team, and they are responsible for creating usable Increment that is delivered at the end of each Sprint.

- Developers are always in charge of planning the Sprint Backlog;
- Adjust the plan every day in line with the Sprint Backlog;
- Act according to Definition of Done;

2.1.3 Scrum Master

The Scrum Master is the person responsible for checking the correct application of the Scrum framework as described in the Scrum Guide. One of his tasks is to help the organization and the Scrum team comprehend Scrum principles.

The Scrum Master supports the Product Owner in various ways, including:

- Finding procedure to effectively determine the Product Goal and lead the Product Backlog;
- Helping the Scrum Team perceive the importance of well-expressed and easily understood backlog items;
- Ensuring stakeholders cooperation when necessary.

The Scrum Master also assists the Development Team:

- Helping the team members being self-managed and cross-functional;
- Building high-value increments;
- Avoiding that the Development team encounters obstacles;
- Guaranteeing the execution of all Scrum events.

2.2 Scrum Events

Each event in Scrum has the goal of inspecting and adjusting Scrum artifacts. The main event that includes all the others is called Sprint. Normally it has a fixed duration of less than one month, because if it were too long the risk and complexity would increase and turn invalid the Sprint Goal.
The Sprint Planning event is a meeting lasting up to eight hours for a one-month Sprint, or less if the Sprint size is smaller. During this meeting the Scrum Master, the Product Owner and the Team come together to determine the work that will be done during the Sprint.
Every day during the Sprint, a communication meeting of the development team is held. This meeting is called "Daily Scrum" and is intended to review the work done, check the progress and adapt the work that needs to be done.
At the end of the Sprint, the Sprint Review meeting, lasting up to 4 hours, is held to inspect the increment and adjust the Product Backlog if necessary. During the Sprint Review meeting the Scrum Team and stakeholders collaborate on what was done during the Sprint.
The Sprint Retrospective is a meeting of up to three hours and is an opportunity for the Scrum Team to examine the past Sprint and create an improvement plan to implement during the next Sprint. The Scrum Team meets for the Sprint Retrospective after the Sprint Review and before the next Sprint Planning (Schwaber & Sutherland, 2020).

2.3 Product Backlog

The Product Backlog is one of the most important Scrum artifacts, it is the representation of the Product Goal. It is a dynamic list of organized items that shows the work that must be done to get the final product (Schwaber & Sutherland, 2020). Normally it consists of functional and non-functional requirements that will deliver the product vision. The Product Backlog is prioritized giving more priority to the items most likely to create value. It is an emergent list, so during the project the Product Backlog could change, some items could be modified or removed, others could be added according to customer needs. Product Backlog items are decomposed into smaller and more accurate items during the Product Backlog refinement. In this phase more details are added, such as a description, order and size of the items. A general view of the structure of the product backlog is provided in Figure 6.

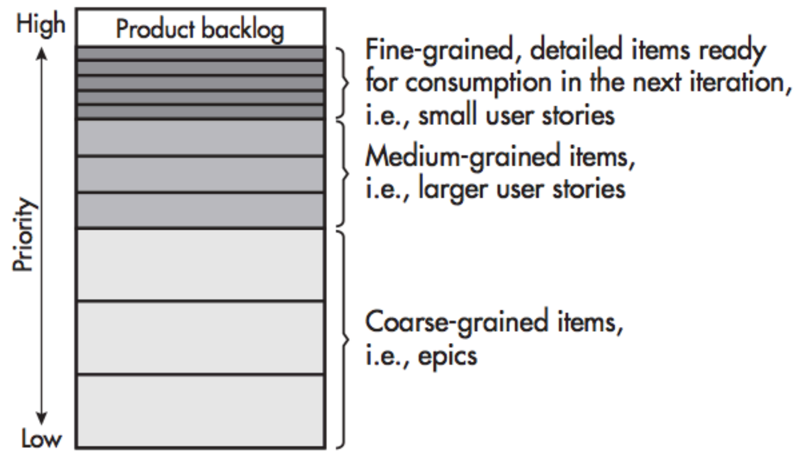

High
Product backlog
Fine-grained, detailed items ready for consumption in the next iteration, i.e., small user stories
Medium-grained items, i.e., larger user stories
Priority
Coarse-grained items, i.e., epics
Low

Figure 6 - Product Backlog (Pichler, 2010)

2.4 Sprint Backlog

The Sprint backlog is the list of work that the Developers needs to complete during the next sprint focusing on the Sprint Goal. This list is generated by selecting a quantity of items from the top of the product backlog determined by what the Development Team believes it can accomplish during the sprint. As knowledge increases, the sprint backlog is updated, and is usually detailed enough so that the team can check their progress in the Daily Scrum.

2.5 Increment

An increment is a tangible step in the direction of the product goal.
Each time an increment is obtained it is added to the previously accomplished increments and the proper functionality of the system is verified.
In order for work to be considered completed, it must meet the "Definition of Done", i.e., achieve the sufficient quality required for the product.
The Definition of Done can be defined by the organization or created by the Scrum Team if it is not present.

2.6 User stories

According to Leffingwell (2011) and Cohn (2004), User stories are a textual notation used more and more often in agile software development, with the aim of acquiring the requirements.
User stories are descriptions that use a simple construction such as “As a ⟨role⟩, I want ⟨goal⟩, [so that ⟨benefit⟩]”. A user story describes a story of a customer or user using the product. In order to get a complete story, it must contain a name, a short narrative and an acceptance criteria that must be verified (Pichler, 2010).
According to Cohn (2004), Scrum does not establish how the product backlog items are delineated, but he prefer to use User Stories. User Stories are normally small and detailed, if they are larger are called epics. Themes usually consists of between two and five coarse-grained requirements (i.e. epics) (Pichler, 2010).

3 PRODUCT PLATFORM DESIGN APPROACHES (TANGIBLE PRODUCTS)

3.1 Product Platform

Due to the growing demands in term of personalized commodities, product platform planning has emerged in order to increase product diversity and Mass Customisation (MC).
McGrath (1995) defines a product platform as a group of common parts, in particular the core technology, used within a wide range of products. Meyer and Lehnerd (1997) define the term as a collection of subsystems and interfaces that constitute a common structure from which a group of products can be efficiently designed and created. Robertson and Ulrich (1998) proposed a broader definition: they describe product platforms as a set of assets (e.g., components, processes, knowledge, people and relationships) that are shared by a group of products.
A product platform can decrease development cost, reduce time-to-market, participate in the extension of product variety and create competitive advantage. This development approach is an effective strategy for addressing mass customization. In recent years, due to the increase in demand for product customization, the product-centered strategy has shifted to a customer-centered strategy, academia and industries have begun to study the product platform to improve adaptability of businesses in an uncertain environment. A platform approach enables efficient customisation, reuse and production standardization.
The platform-based development is present mainly in the automotive industry, where various car models come from the same product platform, but it begins to emerge in many other fields in the domain of tangible products, software and embedded systems.
The Software Product Line (SPL) outlines the development of software product platforms.

According to Zhang et al. (2019), product platform mainly consists of core modules, representing the basic structure of the product family, and variable modules that symbolize the product variety.
Furthermore, the product can be enhanced by improving or redesigning the various modules, therefore is possible to have greater product diversity that is a key factor to achieve mass customisation. Boute et al. (2018) as well affirmed that they manage product variety through the adoption of product platforms as shown in Barco's case, where a product platform which outlines 17 diagnostic displays it was developed.
The platform design model developed by Zhang et al. (2019) is based on product data already available in product lifecycle management (PLM) database, and his model does not only consider the product family, but also product series and could be extended to the whole portfolio.
Product platform planning is crucial to achieve a successful product family with the purpose of increasing competitiveness of the enterprise.
Landahl et al. (2020) use the platform to obtain a wide variety of products to satisfy a large amount of customer needs.
The product platform design process, in the opinion of Cheng et al. (2015), consists of analysing clients' needs, defining Functional Requirements (FRs), associating them to Design Parameters (DPs) in the physical domain and clustering the latter to recognize common platform and adjustable variables, as shown in Figure 7.

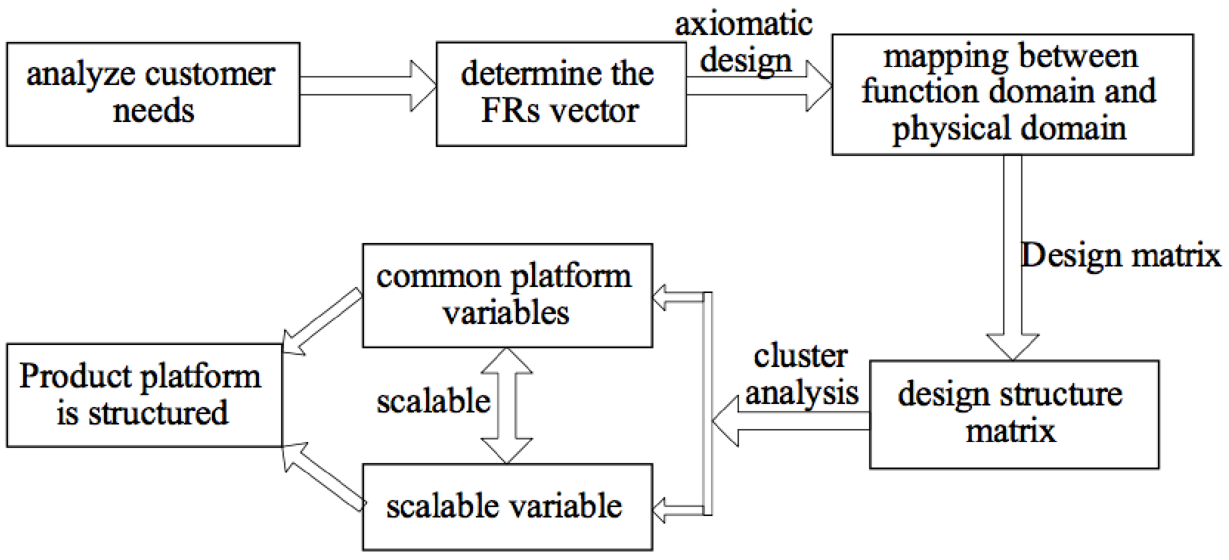


Figure 7 - Product Platform Design process (Cheng et al., 2015)

Product Platforms usually come in three types: modular, scalable, or adaptable. The module based platform consists of interchangeable modules, which are sets of components. The scalable one

permits the adjustment of design variables to obtain product variety. The latter allows both modularity and scalability.
Beyond modularity, it was introduced a new platform element, called Design Asset, that provides efficient customization, reuse and standardization (Elgh et al., 2018; Raudberget et al., 2019). The idea is that a product platform should consist not only of modules or components, but also of information, models, methods and knowledge. In particular, eight domains have been determined with the purpose of creating Design Assets: Process, Product, Synthesis Resources, Analysis Resources, Geometry Resources, Constraints, Solutions, and Projects.

3.2 Methods used in product platforms design:

The following methods, encountered in our bibliographic research, permit us to understand how the product platform is created, how the information is managed and how the product variants take shape. They enable to understand customer's needs in term of features or functional requirements and connect them to physical components.

3.2.1 Quality Function Deployment

According to Akao and Mazur(2003), Quality Function Deployment (QFD) was created in Japan in 1966 to guarantee that customer needs are considered and employed in the development of a new product. The first book dealing with this topic was written by Mizuno and Akao (1978).
Quality Function Deployment (QFD) can be used in the first stage of the product platform planning to detect product features and their importance for the client, as shown in Barco's case, where it is used this method concurrently with the Design Structure Matrix (Boute et al., 2018).
It is normally used to connect Customer requirements to Engineering requirements, and the latter to Components. Furthermore it could be utilize in conjunction with Generational Variety Index (Martin & Ishii, 2002).

3.2.2 Kano model

Kano's model takes the name of its founder Noriaki Kano (1984) and it classifies customer requirements into five categories:

- *Must-be,* basic necessity that the customer expects. Their absence lead to high levels of dissatisfaction;

- *One-dimensional,* attributes that bring to satisfaction when fulfilled and dissatisfaction if they are not fulfilled;
- *Attractive,* unexpected requirements that provide satisfaction;
- *Indifferent,* attributes that are not taken in consideration by the customer;
- *Reverse,* requirements that can provide high levels of dissatisfaction if present.

This model was used in the field of product platform by various authors to understand, classify and prioritize customer needs based on the satisfaction level (Alsawalqah et al., 2014; Cheng et al., 2015) Pohl et al. (2005) used a Kano-method Portfolio Planning to develop a customer-oriented Software Product Line (SPL). Pichler (2010) also outline the importance of this method, applied to the product vision and to the product backlog, in the Agile product management with Scrum.
Wu & Wang (2012) proposed a Fuzzy extension of Kano's model to deal with uncertain Customer Requirements.

3.2.3 Axiomatic Design

Axiomatic design (AD) was proposed by Suh and Sekimoto (1990) and permits to connect customer, functional, physical and process domain. This method links Functional Requirements (FR) and Design Parameters (DP), obtaining a hierarchical representation, as shown in Figure 8. Through an iterative zigzagging process, the FR – DP connection is established and represented by the design matrix (Cheng et al., 2015).
As also reported by Johannesson (2017) this method shows the link between functional requirements (FRs) and design parameters (DPs). Furthermore the zigzagging process denotes that before decomposing a functional requirement into other requirements it is necessary to determine intermediate solutions (i.e. DPs).

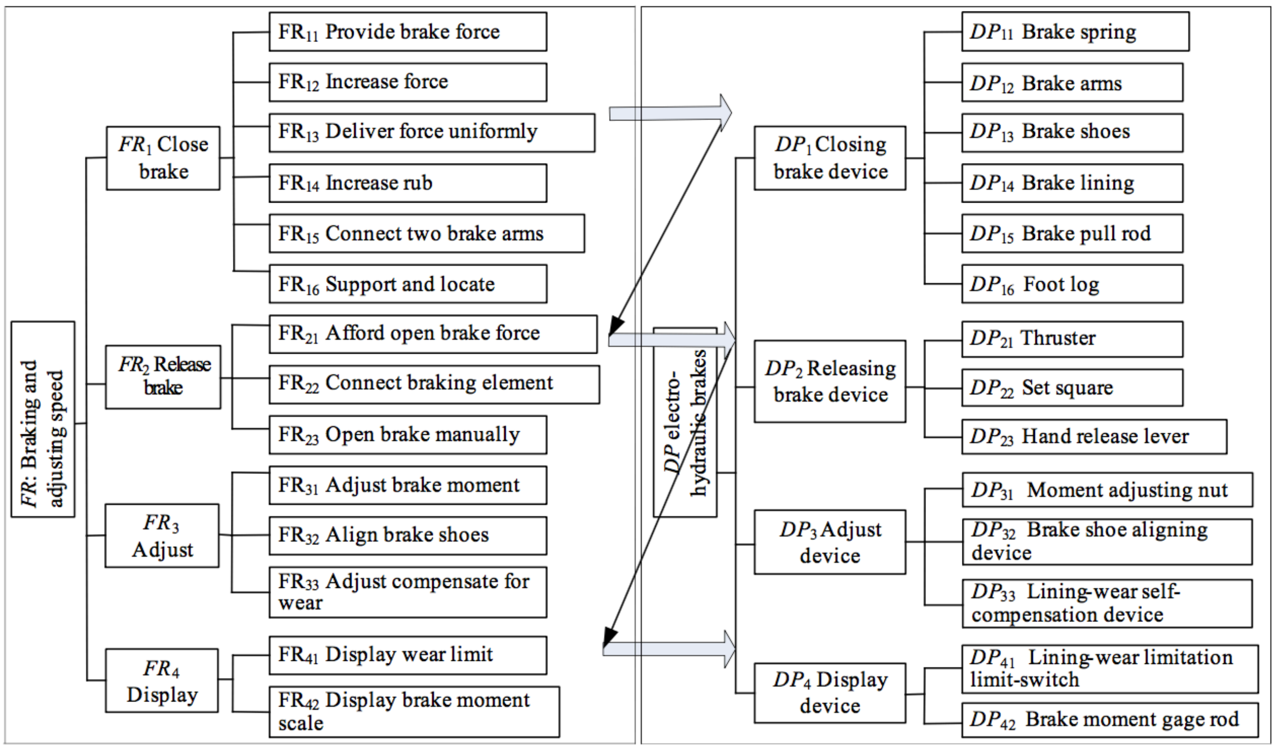


Figure 8 – FR – DP zigzag mapping based on Axiomatic Design (Cheng et al., 2015)

3.2.4 Design Structure Matrix

Donald Steward (1981) first coined the term Design Structure Matrix and implemented this concept to manage the design of complex systems.

The Design Structured Matrix (DSM) is normally used to show interconnections in the physical domain with the purpose of decompound the system in substructures that could be considered as modules. Cheng et al. (2015) utilize the DSM to represent the relationship among the Design Parameters. He developed the Design Structure Matrix (DSM) from the Design matrix, obtained in turn from the Axiomatic Design process, as shown in Figure 9.

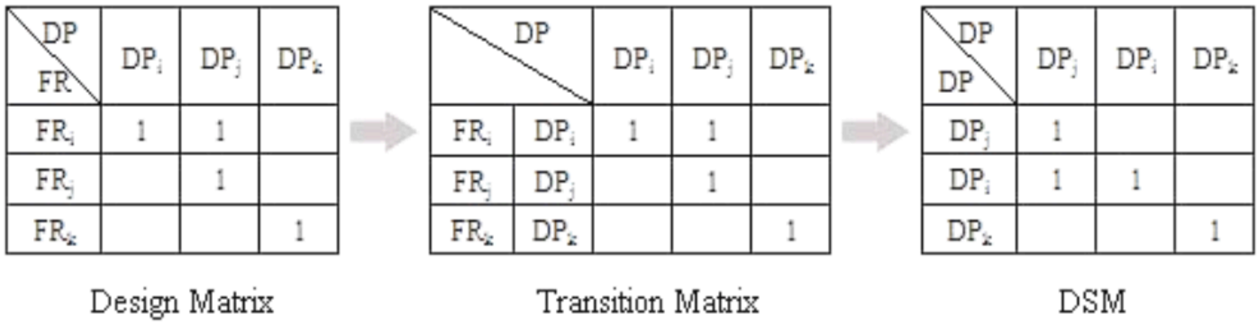
According to Cheng et al. (2015) a limitation of this method is that is suitable only for products that have already been designed.

Simpson et al. (2012) use the DSM to show a generic product architecture enhanced with a component change propagation grade (Low – Medium – High) which exhibit if an alteration of a specific component affects another one.

According to Johannesson et al. (2017) DSM can be used to examine system alternatives combined with axiomatic design matrices, and trade-off curves.

Boute et al. (2018) also used the DSM in their approach along with Quality function deployment to explore the features, their importance for the client and their connection with element in the Bill of

materials (BOM).
Baylis et al. (2018) use this method to show the connection between components, to group them into modules. The DSM permits also to see if components of different modules are linked, so these modules could be merged in a building block.



Design Matrix

FR \ DP	DP_i	DP_j	DP_k
FR_i	1	1	
FR_j		1	
FR_k			1

Transition Matrix

DP		DP_i	DP_j	DP_k
FR_i	DP_i	1	1	
FR_j	DP_j		1	
FR_k	DP_k			1

DSM

DP \ DP	DP_j	DP_i	DP_k
DP_j	1		
DP_i	1	1	
DP_k			1

Figure 9 - Design Structure Matrix process (Cheng et al., 2015)

3.2.5 Pruning Analysis and Attribute matching

Zhang et al. (2019) suggest a new approach consisting of two methods, i.e., pruning analysis and attribute matching. The first one allows to represent the core structure of the product platform discovering common parts of several product families. A Minimum Structure tree and a Residual Subset are created as a result, representing respectively the Basic framework and the Variables.
The latter permits to group product modules into four types based on their sharing degree. In particular, the product modules are classified as Private (PrM), if the product module is included exclusively in a specific product family, Families sharing (FSM), if it appears in several product families, Series sharing (SSM), if it is part of every family of the product series, or Global sharing (GSM), if it is contained by each product series of the enterprise. The SSM and GSM groups are part of the Basic product modules and they are derived from the Minimum structure trees. The PrM and FSM are Variable product modules and they originated from the Residual subsets. (Zhang et al., 2019)

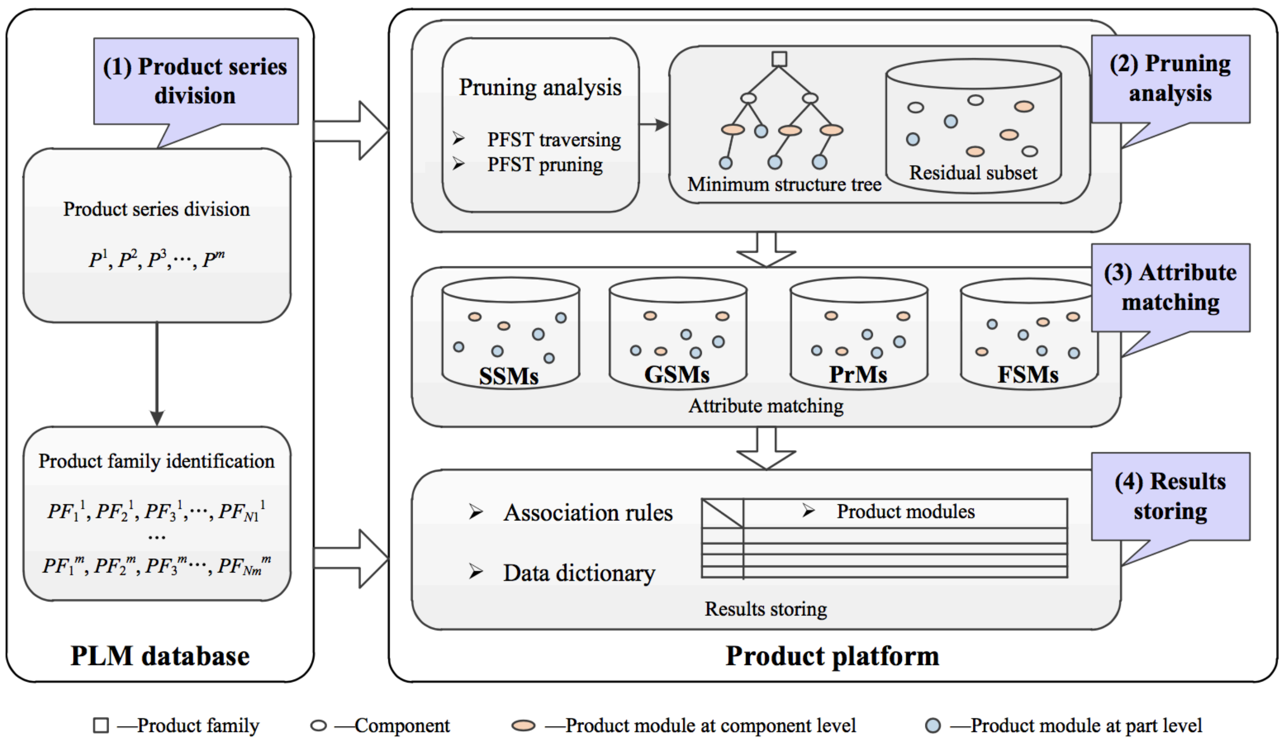
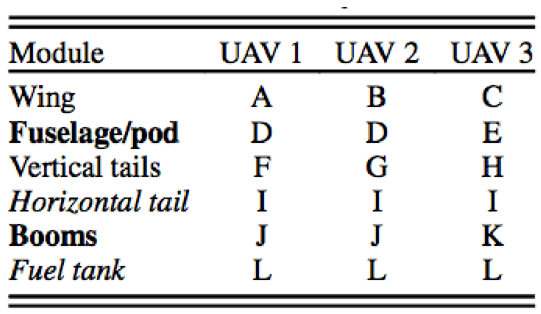


Figure 10 - Product Platform planning (Zhang et al., 2019)

3.2.6 Commonality matrix

Chowdhury et al. (2011, 2016) used the commonality matrix in their mathematical approach, called Comprehensive Product Platform Planning (CP^3) Framework, with the aim of designing a product platform for universal electric motors and unmanned-aerial-vehicle (UAVs).

In the original CP^3 model the commonality matrix was represented by common physical design variables between products (Chowdhury et al., 2011), while the improved framework consisted of module-sharing variables (Chowdhury et al., 2016). According to Chowdhury et al. (2016) this method supply a product-platform plan, shown in Figure 11, which shows the sharing of modules and features and permits the generation of viable platforms. Furthermore it can be used in both cases of modular or scalable attributes. The Figure 11 shows that two modules are totally shared between the family of products, i.e. Horizontal tail and fuel tank, and other two are partially shared, i.e. Fuselage and Booms, in the Pareto solution with maximum commonality.
The commonality matrix shows if a design variable is required, not relevant or optional for a specific product. The Product Platform Plan resulting allows to organize the design variables in Platform, Sub-platform and Nonplatform, respectively if the variable is shared by all the products in the family, only with a group of products, or with any product. (Chowdhury et al., 2011)



Module	UAV 1	UAV 2	UAV 3
Wing	A	B	C
Fuselage/pod	D	D	E
Vertical tails	F	G	H
Horizontal tail	I	I	I
Booms	J	J	K
Fuel tank	L	L	L

Figure 11 - UAV Platform Plan (Chowdhury et al., 2016)

3.2.7 Mixed Integer Linear Programming (MILP)

Mixed integer linear programming is a subset of linear programming, founded by Kantorovich(1939). ElMaraghy and Moussa (2019) use a mixed integer linear programming (MILP) model to create the product platform of a guiding bushes family. The optimization problem recognizes the core features that establish the platform and other characteristics that can be added or removed to obtain the product variants. The product derivation is obtained through additive or subtractive processes (ElMaraghy & Moussa, 2019).
Also Chowdhury et al. (2011) use MILP model to obtain the optimization of the Electric motors family.

3.2.8 Clustering algorithm

According to Zhang et al. (2019) the clustering algorithm is the most frequently used algorithm to identify the sharing parts in scalable product platform planning.
Cheng et al. (2015) use clustering algorithm to distinguish cluster of design parameters (DPs) and this allows to identify common platform and scalable variables.
The DPs having the higher effect on other design variables are gathered in the same cluster and will become part of the common platform variables. Therefore, the others are designated as adjustable variables. (Cheng et al., 2015)

3.2.9 Generational Variety Index (GVI)

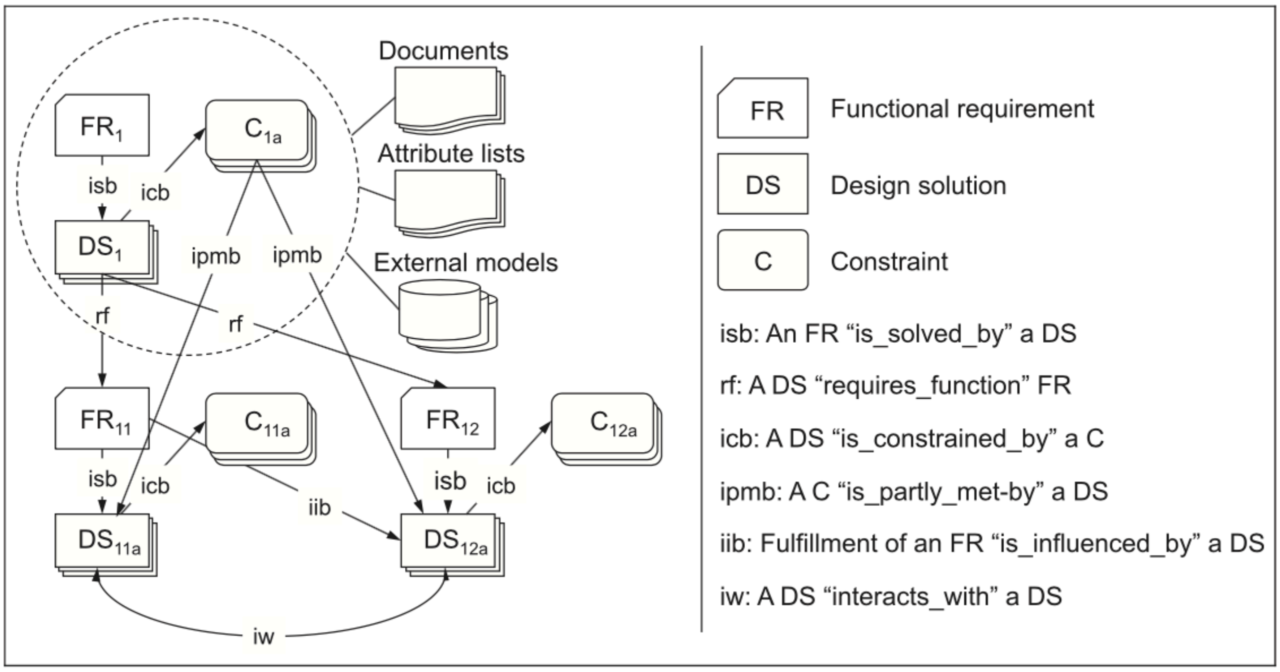
The generational variety index (GVI) helps identify the components of product variants that are more likely to require redesign in the future (Martin & Ishii, 2002).

Customer requirements are normally classified in “High - Medium - Low” where High implies that the requirement evolve quickly and Low means that not much redesign is needed in the future (Martin & Ishii, 2002).
For this reason components with low GVI values are considered potential platform elements.
According to Simpson et al. (2012), it is advisable to modularize components with high GVI so as to be more easily exchanged or improved.
Li et al. (2016) also employed Variety Index (VI), together with Fuzzy arithmetic and Change Propagation Index (CPI), to classify group of modules as standardized or flexible. Components of a flexible module are subsequently categorized as common or scalable through Lifecycle factors such as Design complexity, Sensitivity of manufacturing cost and Assembly complexity.

3.2.10 Enhanced function-means tree (EF-M)

The model was originally developed by Tjalve (1979, p. 9) and represents a hierarchical decomposition of the principal function into sub-functions and means to accomplish these.
The function-means (F-M) tree can be considered comparable to the Axiomatic Design because they both decompose the system showing the interconnections between Functional Requirements and Design Solutions (Johannesson et al., 2017).
The original function-means tree was improved by Schachinger & Johannesson (2000) by considering additional parameters like Constraints (Cs), defined as non-functional requirements.
The introduction of the Constraint variable in the Enhanced function-means (EF-M) tree, make clear why a DS is selected and why other options have been refused.
Furthermore Johannesson et al. (2017) combined the EF-M tree with the Configurable Component (CC) model proposed by Claesson (2006) to allow modularity and scalability.
Landahl et al. (2020) used the EF-M tree to describe product variety of a Turbine Rear Structure. The reason why this method shows product variety is due the fact that when it is merged with CC model, it permits modularity through compatible solutions and scalability through changeable parameters within fixed range. Moreover, in Landahl et al. (2020) opinion, this method allows variety at product level, production resources level and production process level.
According to Johannesson et al. (2017) their approach allows building the platform from the beginning and also adding new FR, if requested. In this case a DS to satisfy that FR will be found and evaluated in iterations. In the Figure 12, drawn by Levandowski et al. (2014), is possible to understand the interconnection among different assets provided by the EF-M tree that provides the Design Rationale (DR). The Configurable Component tree shown in Figure 13, drawn by Levandowski et al.

(2015), allows to have an insight of the product architecture. Furthermore, it shows the possibility to obtain scalable or interchangeable Design Solutions (DSs) by shaping FRs. Moreover, DSMs, Axiomatic design and trade-off curves are used to examine various architectural options. Johannesson et al. (2017) normally analyze pre-embodiment solutions to avoid creating 3D models that take a long time to be developed.



Documents
Attribute lists
External models
FR1
C1a
DS1
isb
icb
ipmb
ipmb
rf
rf
FR11
C11a
FR12
C12a
isb
icb
isb
icb
iib
DS11a
DS12a
iw
FR Functional requirement
DS Design solution
C Constraint
isb: An FR "is_solved_by" a DS
rf: A DS "requires_function" FR
icb: A DS "is_constrained_by" a C
ipmb: A C "is_partly_met-by" a DS
iib: Fulfillment of an FR "is_influenced_by" a DS
iw: A DS "interacts_with" a DS

Figure 12 - Enhanced Function Means EF-M tree (Levandowski et al., 2014)

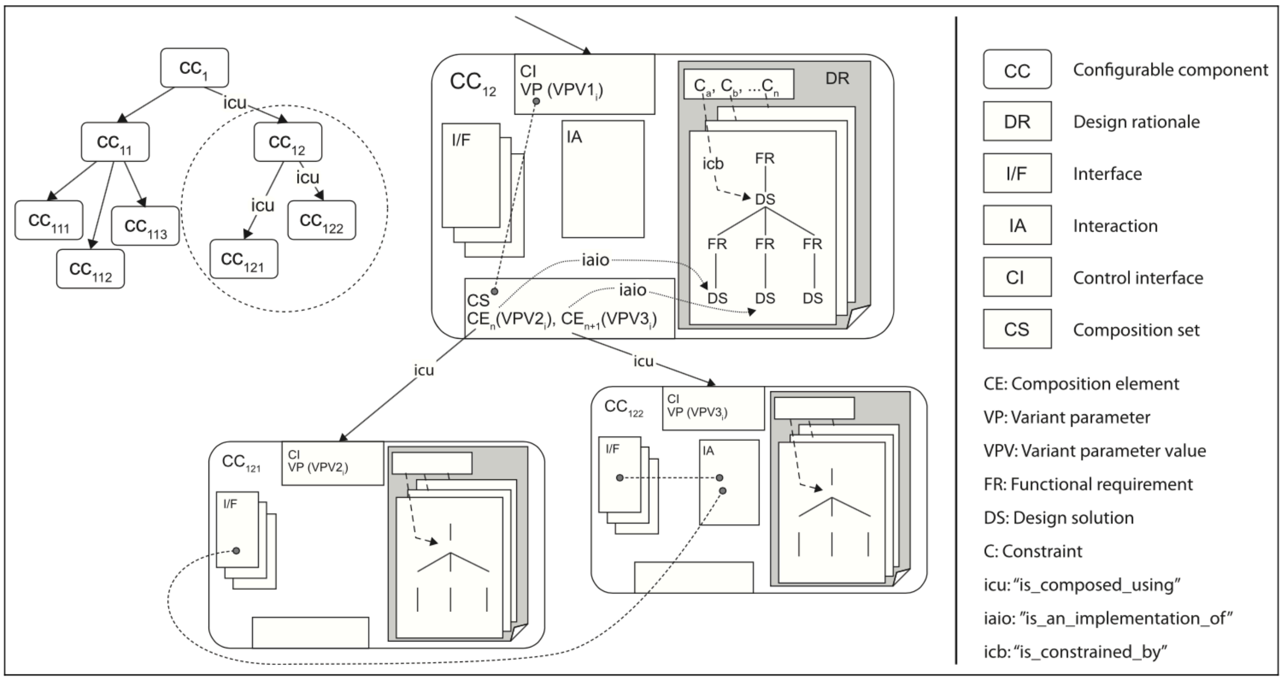


Figure 13 – Configurable Components tree (Levandowski et al., 2015)

3.2.11 Similarity index and Sensitivity index

The similarity index, a well-known principle in the product architecture literature, points out the grade of similarity between two physical components (Kamrani et al., 2002). Two elements should be placed together in case of high similarity index, and if one element is part of the platform, the related component should also be added. The sensitivity index reveals the alteration of a physical component's value based on differing weights of the customer requirements (Kim et al., 2006).
Kim et al. (2006) use these two methods to understand which components should be part of the product platform. In particular are considered platform elements those who have wide similarity index and low sensitivity index. The similarity index were used also by Cheng et al. (2015) in the clustering analysis together with the Design Structure Matrix (DSM) thus creating a Similarity Matrix.

Year	**Product**	**Platform type**	**Method**	**Information**		**Autor**
				Requirements level/Functional level	**Physical level**	
2015	Electro-hydraulic drum brakes	Scalable	Axiomatic Design - Design structure matrix - Clustering analysis	Functional Requirements	Design Parameters	Xianfu Cheng, 2015
2017	Valve	Modular	Pruning analysis - Attribute matching	Product Modules (Function parameter)	Product Modules (Structure parameter)	Qiuhua Zhang, 2017
2018	High-tech medical displays	Customizable (monolithic)	Quality function deployment - Design-structure matrix - Optimization model (Cost model)	Features - Costs	Design Parameters	Robert N. Boute, 2018
2011	Universal electric motors	Modular and Scalable	Mixed integer nonlinear problem - Platform segregating mapping function - Particle swarm optimization algorithm - Commonality constraint matrix - Cost decay function	-	Design variable	Souma Chowdhury, 2011
2016	Unmanned-aerial-vehicle	Modular	Commonality matrix	Performance	Modules = Multiple design variable	Souma Chowdhury, 2016

2019	Guiding bushes family	-	Mixed integer linear programming (MILP) model	Costs	Features and relationships	Hoda ElMaraghy, 2019
2012	Unmanned ground vehicles (UGVs)	-	Generational Variety Index (GVI) - Design Structure Matrix - Commonality specifications - Multi-objective optimization	User needs/requirements	Commonality specifications	Timothy W. Simpson, 2012
2017	Aero engine sub-systems, Vehicle seats, Electromagnetic contactors	Modular and Scalable	Function-means models - Configurable Components System model - Set-based concurrent engineering processes	Functional Requirements	Design Solutions - Constraints	Johannesson, 2017
2015	Rear frame of a jet engine	Adaptable(Modular and Scalable)	Configurable component - Function–means tree	Functional Requirements - Performance parameters	Design Solutions - Constraints	Levandowski, 2015
2020	Aero engine sub-systems	Modular and Scalable	Function-means model - Set-based concurrent engineering processes - Production operation model	Functional Requirements	Design Solutions - Constraints	Landahl, 2020
2006	Speed reducer	-	Similarity index - Sensitivity index	Customer requirements	Physical elements	Kim, 2006
2015	Spraying machine	Adaptable(Modular and Scalable)	Variety index - Change propagation index - Fuzzy alogrithm	Functional modules	Product attributes	Li, 2015
2018	impact drivers and electric drills	Modular	Product component matrix - Pareto front of maximum commonality - Design Structure Matrix	Clustering cost	Components	Baylis, 2018

Table 1 - Product Platform Design Approaches

3.3 Information managed

In most of the cases encountered, shown in the Table 1, it was noted that the product functional requirements were the basis for the construction of the product platform in the functional domain. Instead, others authors considered physical features as shown in Elmaraghy (2019).
As a rule, the functional layer is subsequently connected to the physical domain, represented mainly by Design parameters, Design variable, Design solutions or Physical elements.

Boute et al. (2018) consider each product of the portfolio as a group of features and the platform as a collection of design parameters.
In Barco's case the platform consists of design parameters, as power consumption or pixel pitch, that can satisfy product features, as megapixel count or colour levels (Boute et al., 2018).
As shown also in Van den Broeke et al. (2017) the set of the possible platforms can be represented by a platform-product tree, where the amount of design parameters points out the profundity and the variants of each design parameters outline the wideness.
Cheng et al. (2015) structure the product platform of electro-hydraulic drum brakes on functional requirements (FRs) linked to design parameters (DPs) in the physical domain. They investigate customer needs and separate product functional requirements resulting in basic, expectable and adjunctive through Kano model. In particular the basic requirements are represented by functions, as "braking" and "brake release", that are hierarchically decomposed and connected to design parameters through the axiomatic design. Design parameters are also broken down at the same time, so that each FR corresponds to a specific DP. For example, the FR1 "close brake" is decomposed in the FR11 "provide brake force" and they are respectively represented by the DP1 "closing brake device" and DP11 "brake spring". The DPs relations are shown in the Design structured matrix and by means of the clustering analysis they are classified in common platform, controllable and customization parameters (Cheng et al., 2015).
Zhang et al. (2019) consider the product module, that it consists of three parameters, such as structural, functional and procedural, to build the product platform of power-driven and self-driven valves. According to them the structural and process parameters are mutable attributes, instead the function remains fixed. For this reason, in the Attribute matching phase, they classify the product modules based on function parameters such as "seal", "link", "flow control", "pressure control" etc.. The product families are represented in Product family structure trees (PFSTs), and product modules are classified in four group based on the sharing level among product families or product series as shown above.
Chowdhury et al. (2011) in their first model considered the design variables in the platform planning of ten universal electric motors. The product family is represented by Functional requirements like Torque specifications, Design constraints as output power, total mass or efficiency, and Design variables as Number of turns on the armature, radius of the motor or thickness of the stator. Through the commonality matrix the design variables are sorted in Platform, Sub-platform or Nonplatform depending on whether they are part of several products, of a specific group of the product family, or if they are not present in more than one product (Chowdhury et al., 2011).
In the enhanced model, Chowdhury et al. (2016) considered group of modules to obtain a

reconfigurable Unmanned Aerial Vehicles, where a module can be seen as a set of design variables. The modules were mostly physical parts of the product like wing, vertical tail, horizontal tail, fuel tank and fuselage, everyone composed at least by one design variable. For example the fuselage was expressed by cross section and length. In this model the commonality matrix considers also a module-inclusion variables.

Simpson et al. (2012) in their approach to design a family of unmanned ground vehicles (UGVs) consider User requirements like “weight, speed, range, lift capacity” in the product plan phase.

A generic UGV architecture is represented by a Design structured matrix which display sub-system/component interconnections and how each component affects another one.

The GVI analysis compare performance requirements as “range, slope climb, Maneuver width” with sub-systems as “chassis, battery, tracks, cameras” to understand which component can be shared in the product family. The analysis also concerns about Design parameters as “length, width, height” in the case of Chassis subsystem, showing common parameters between two or more vehicles. Furthermore, the mathematical model provides a complete structure diagram of the System decomposition (Simpson et al., 2012).

One of the broader methodologies that permit to represent the product in all its levels, is represented by Johannesson et al. (2017). They combined two methods to obtain a broader view of the product family. The first method is the Enhanced function-means (EF-M) tree that expresses the relation among three parameters: FR – DS – C.

In the specific, the FR describes a certain purpose of the product or a subsystem, DS is the physical solution that allows to accomplish the FR, and C is a requirement that indicates DS’s limitations.

The EF-M tree express interconnections at any hierarchical level, including system dependencies that are utilized in the structure analysis through Design structured matrices (DSMs) and Axiomatic design (AD) matrices.

The second method is the Configurable Component system model that illustrates an entire family of alternatives. The CC system model contains the Design rationale (DR) and the EF-M tree. Design rationale (DR) is a set of information that clarifies why the product or the component was designed in that way including all the reasons and options analyzed (Gedell & Johannesson, 2013). Furthermore other variables are included as Variant parameters (VPs) and Control Interface (CI) that allow to build suitable variants.

According to Johannesson et al. (2017) the CC model permits modularity by substituting CC or DS, and scalability due to the fact that CC and Ds can be altered within a range expressed by parameter bandwidth. Johannesson et al. (2017) applied their method on aero engine sub-systems, vehicle seats and electromagnetic contactors. Also Landahl et al.(2020) used the EF-M model for a Turbine Rear

Structure (TRS) and we have FR like “convey thermal loads” that can be settled by DS as “cooling system” or “heat shield”. Moreover, constraints, product resources and product operations are considered.

In the speed reducer platform of Kim et al. (2006) are considered physical components as “gears, shafts and bearings” and customer requirements as “size, weight and cost”.

Li et al. (2016) consider hierarchically modules, components and parameters. In particular they connect Modules and Functional Attributes as “Spray capacity, Power or Pressure” into an Attribute-module matrix. Modules are clustered into standard or flexible, the latter are decomposed in components like “Hooper, batching plate or rotor” and classified into common or scalable based on lifecycle factors.

Baylis et al. (2018) take into account components such as “Clamshell, armature, stator or rotor” clustered into modules that are grouped into building blocks.

Few of the cases shown considered mainly the physical sharing degree to build the product platform. Instead, in most of the cases seems the product platform is built based on functional requirements, which translate the customer needs, and a design solution is found to satisfy them.

3.4 Product derivation

According to Cheng at al. (2015), product variants can be generated adding, eliminating or exchanging platform modules. In a different perspective, ElMaraghy & Moussa (2019) sustain that is possible to originate the product variants through additive or subtractive processes.

In the case shown by Boute et al. (2018) the different products can be derived summing extra elements to the product platform.

3.5 Product improving

According to Zhang et al. (2019), redesigning the PrM, i.e. product module not shared by various product families, or adding new product modules lead to a variant development for the product family. An alternative to modify various product families simultaneously is to redesign the Family sharing product module (FSM). According to Johannesson et al. (2017) it is possible to improve the product platform adding new functional requirements. New design solutions to satisfy these requirements are created and analyzed.

3.6 Agile physical Product Platform

The only case study dealing with the application of agile methodologies within physical product platforms is the one shown by Varl et al. (2020). This study propose a method to of power transformers but it is not providing a depth explanation and seems like the Scrum agile methodology has been applied only at the team level. Furthermore, Scrum artifacts, as Product backlog, are not present.

4 SOFTWARE PRODUCT LINE SCOPING APPROACHES

Software product line Engineering is a Software development concept that combines platform-based development and mass customisation (Pohl et al., 2005). The main difference between Software product line and single product development is the need of two separate development processes and the necessity to define the variability. The purpose of scoping approaches is to identify more important features and products that will be part of the product family. A key section of the product line scoping is the commonality and variability analysis. Common and variable features can be identified through the Kano method (Pohl et al., 2005). Variability can be defined through use case models, feature models, message sequence diagrams and class diagrams.
Another method used to define the variability is the Orthogonal Variability Model, usually combined with the Feature model with the aim of avoiding the feature tree overloading.
The use of explicit documentation, as the Feature model and the Orthogonal Variability model, provides some advantages. These models improve decision making, communication with customers and traceability of variability. The traceability permits to obtain the reuse of assets.
The SPL development process is split into two phases called Domain Engineering (DE) and Application Enginerring (AE), as shown in Figure 14. The first one aims to create a product platform and define common and variable assets. The second phase wants to derivate products starting from the product platform found in the previous phase. Kano model is generally used in the commonality analysis to understand the requirements importance. In particular, are considered Common part the Basic requirements and the High-priority requirements for a large group of customers.
The PuLSE-Eco V2.0 approach, proposed by Schmid (2002), divided the scoping phase in three main components. The first is the Product line mapping, an high-level domain analysis, which aims to obtain a description of the product line, including relevant features, by combining existing information regarding the planned product portfolio, existing systems, available product plans, expert knowledge (from interviews). As a result of this phase an initial Product map is created. The second

phase, called Domain potential assessment, analyzes benefits and risks related to the product line development. The last phase is the Reuse infrastructure scoping aims to identify reusable assets.
Alsawalqah et al. (2014) proposed an approach to optimize the scope of a Software product platform. In the customer needs analysis phase Kano model and Quality Function Deployment Product Portfolio Planning (QFD-PPP) are used to comprehend and organize customer requirements. The latter are prioritized using the Kano's absolute importance value, using the impact on customer satisfaction (SATj) e dissatisfaction (DISj). Furthermore, an integer linear programming problem is generated to optimize the scope.
One of the most recent methods called CoMeS was presented by Ojeda et al. (2018). This approach identifies roles, exhibit tangible artifacts and shows a series of steps to build the scope. A Detailed list of steps to construct the product map and the correlated artefact are shown. In this part features are classified as essential, desirable or inconsistent with the product. This method allows to understand the relation within artifacts and their inputs and outputs making the scope easy to understand for all the stakeholders. An example of the artifacts interconnection is shown but there is a lack of a case study. Another case of a Pick-and-Place Unit (PPU) is shown by Hinterreiter et al. (2020), in which Feature-oriented development and variation control system are used. In particular the method consists of a feature model defining feature commonality and variability; a variation control system that connects features to implementation artifacts; and product configurations, that consists of code and artifact. There are two types of approach: Proactive and Reactive. The first one is like the waterfall approach to conventional software, with analyze, architect, design, and implement all product variations on the foreseeable horizon up front. The reactive approach build the core assets incrementally based on products built as single-systems.

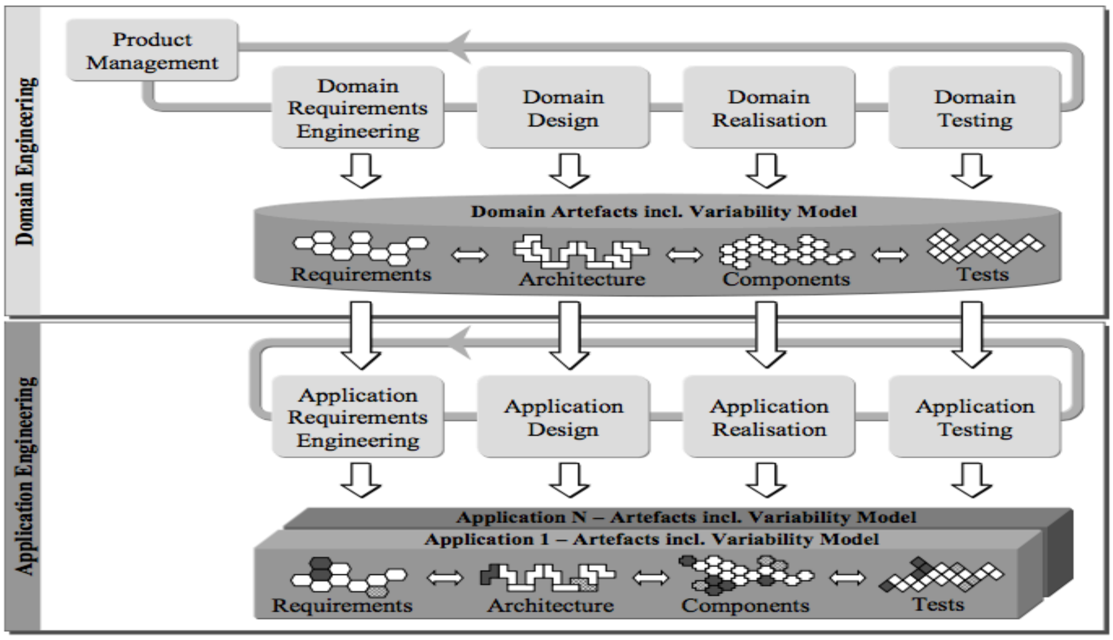


Figure 14 - Software Product Line Engineering framework (Pohl et al.,2005)

4.1 Feature Model

Feature model is one of the most used methodology that permits to represent the entire product family in software product line development. This method was conceived by Kang et al. (1990) in the Feature-Oriented Analysis (FODA). Feature models have been used to express the high-level requirements of an architecture (Pohl et al., 2005).
This model is largely employed to manage common and variable assets in software product line and plays a key role in customization. It consists of features hierarchically organized in a tree structure (Benavides et al., 2010). This model is not only used in software industry, but also in other fields as automotive (Oliinyk et al., 2017). Abrantes & Figueiredo (2014) presented an analog-to-digital converters (ADC) case study showing how the feature model can be a useful method for scoping the New Product Development Portfolio. This demonstrate that can be also applied to physical products domain. The feature model is intuitive and permits to perceive graphically if features are mandatory, optional and alternatives, as shown in Figure 15. Furthermore, the Extended feature model can express complex constraints as *''If attribute A of feature F is lower than a value X, then feature T cannot be part of the product''* (Benavides et al., 2010).

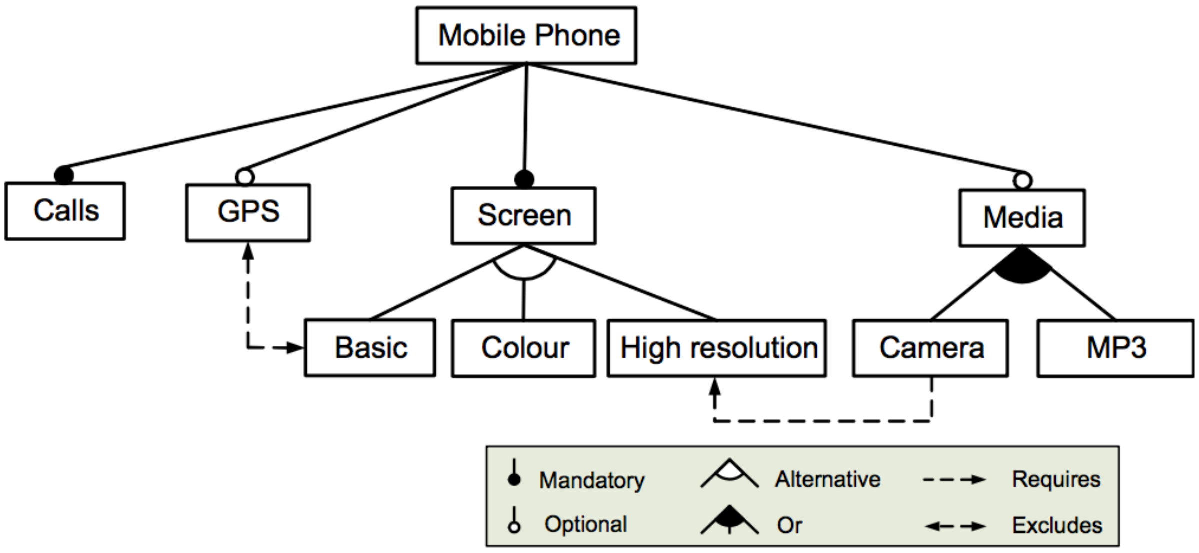


Figure 15 - Example of Feature Model (Benavides et al., 2010)

4.2 Orthogonal Variability Model

To avoid overloading the feature model with variability information, the latter are normally defined by another method, called Orthogonal Variability Model, shown by Pohl et al. (2005). In this way, the variability definition is more clear and misinterpretations are avoided. The domain variability model defines the variability of the software product line through the use of variation points. The variation point applies to all kinds of development artefacts, i.e. requirements, architecture, design, code, and tests. Additional information are added to specify why the variation point was introduced. For example in case the product is sold in different countries and have different stakeholders needs. “Variation point”, “Variant” and “Variability dependencies” are the key elements of the model. This model shows the dependency between the variation point and the variant, which can be Optional or Mandatory. It permits also to select a minimum or a maximum number of variants, and shows the constraints.

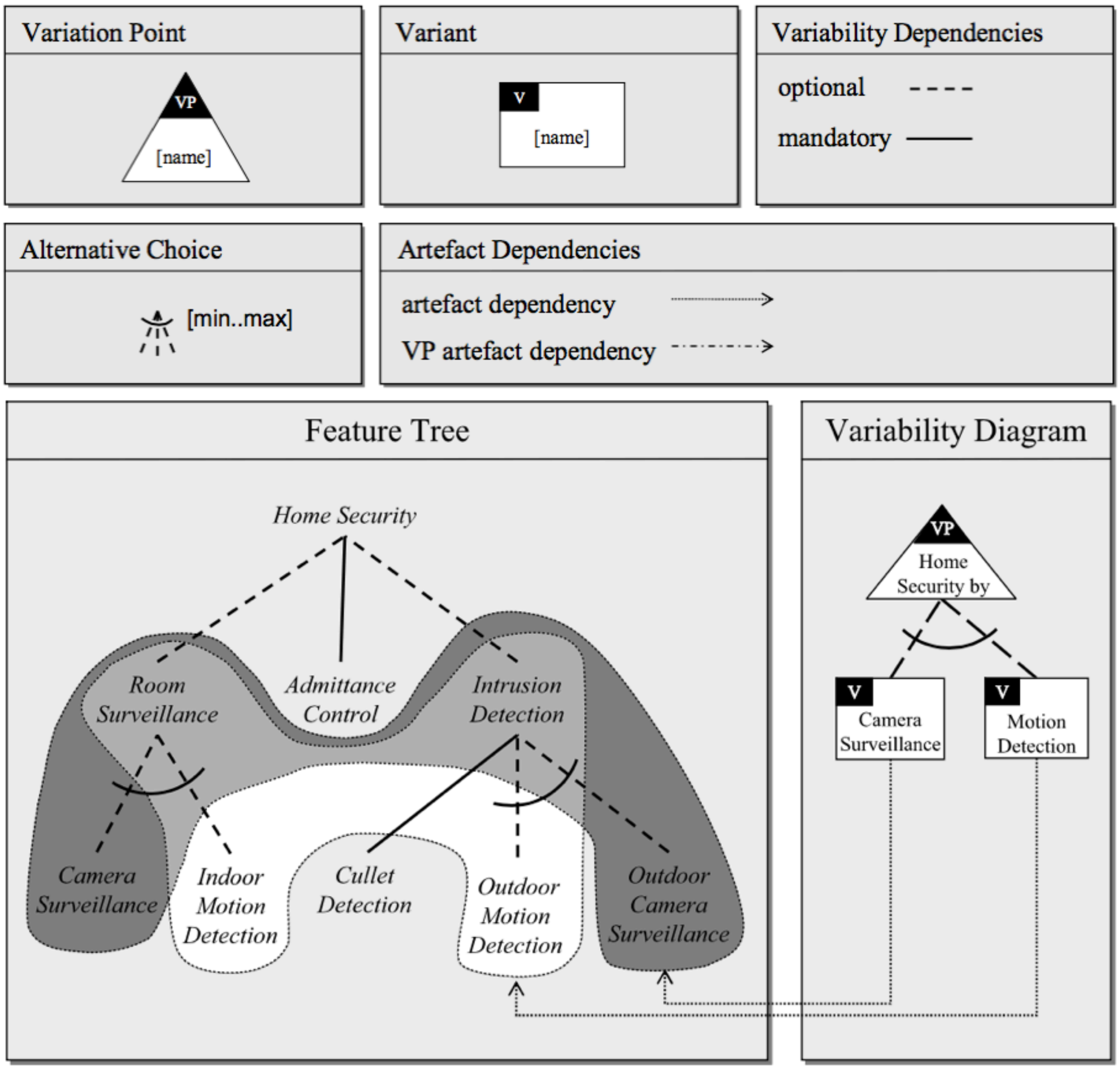


Figure 16 - Orthogonal Variability Model (Pohl, 2005)

5 AGILE SOFTWARE PRODUCT LINE SCOPING APPROACHES

The connection between Agile Software Development (ASD) and Software product line (SPL) is getting bigger and as reported by Da Silva (2012) there are proofs of the use of agile methodologies in the SPL activities, such as planning game, incremental design, Extreme Programming (XP), Scrum, Test Driven Development (TDD), collaboration engineering, Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Lean and Evo.

A new approach called Agile Product Line Engineering supports the combination of Software Product Line Engineering and Agile Software Development with the purpose of reducing the big upfront design. Another goal is to make the development of software product lines more flexible and adaptable to changes. Although it is a promising approach, designing and evolving the product platform, meeting the agile principles, it is challenging (Díaz et al., 2014).

Klünder et al. (2018) show the outcomes of a document review and proposed a method called Agile hamburger for large companies without going into the scoping phase.

Some cases of hybrid approaches that attempt to combine agile methodologies within SPL are given below.

5.1 AgiFPL

Haidar et al. (2017, 2019) proposed an Agile Framework for managing evolving Product Line called AgiFPL, constituted of Goal-oriented requirement engineering (GORE) approach, Feature modelling, Scrumban in Domain Engineering (DE) processes and Scrum in Application Engineering (AE) development processes. The GORE framework report stakeholders aims and the feature model show the product line variability. In the first place the goal model is prepared and turned into a feature model, afterwards User stories are produced (Haidar et al., 2019).

User stories are explanations of a feature consisting of Format, Role, Means, Ends as *"As a ⟨role⟩, I want ⟨goal⟩, [so that ⟨benefit⟩]"*. As an example, the User story of the feature "Invoicing" is *"As ⟨Accountant⟩, I want to ⟨Generate and Send Invoices⟩, so that ⟨the Invoice can be paid⟩"*.

In the Domain Design (DD) phase a reference architecture is generated, commonalities and variabilities are identified and Feature models are created to determine Feature Backlog items. Features are classified as User stories (US) and Domain experts create the Selected Backlog (SB), a list of tasks that the Development team need to complete next. Whenever a US is introduced into the SB or into the production flow a planning session is executed. The production flow is composed by

task backlog, task in progress, task done, story testing and story done. When the increment is prepared and approved, the feature is positioned in a Common assets warehouse.
In the Application Engineering (AE) phase product owner and stakeholders are more present and when a new feature is insert the requirement phase is performed. As a result, features that appear in the database are classified as Selection of feature (SoF) otherwise if they do not exist are categorized as Definition of feature (DoF). App Backlog is built and the work is accomplished in Sprints. In particular, when the product owner (i.e. "App i Owner") has new aims, the "Line i Team" can follow the domain engineering procedure to create new reusable artefacts that are not present in the warehouse. Otherwise, if the new objective do not influence the product line but just an individual product, and there are not similar features in the common assets, then User stories and Backlogs are immediately generated.
Furthermore Haidar et al. (2017) show a small case study of an ERP platform to implement features as "Invoicing Application" and "eTracking service".

5.2 APLE method

One of the most recent approaches was suggested by Kiani et al. (2019, 2021). They have conducted research on various existing approaches and proposed a new Agile Product Line Engineering (APLE) method, shown in Figure 19, which intends to combine the strengths of Software product line (SPL) and Agile Software Development (ASD). This approach is built on the Scrum agile framework and the Scoping phase is not kept apart as in the conventional SPL, instead the artifacts are defined, built and/or upgraded according to the necessity. This is a reactive manner coherent with the agile principle YAGNI (You aren't Gonna Need it) and in opposition to the conventional proactive SPL approach.
The Product Owner (PO) is the stakeholder representative, as in the Scrum method, and create the Feature Catalog composed by User stories (US) and Acceptance Tests (ATs). The latter are procured within a database called Info Base, if nothing is found a new Product Line is begun otherwise the Requirement Classification starts. The sorting phase is based on a resemblance value and as a result New, Variable and Core requirements are obtained respectively if the value is lower, higher or equal to the threshold value. New requirements are included in the Product Backlog. Variable requirements represent components that can be upgraded or refined. If the customer requirement is incompatible with the component a new component is developed, otherwise the component is examined to understand if satisfy the new requirement. Core requirements are reusable and are included in the SPL backlog if they are not already implemented. Stories can move from SPL backlog to product backlog and the other way around. There are two types of teams, Application Engineering (AE) team

and Domain Engineering (DE) team. The first one focuses more on product derivation from the PL architecture, while the second one allows to shape the product architecture in a iterative and incremental way.

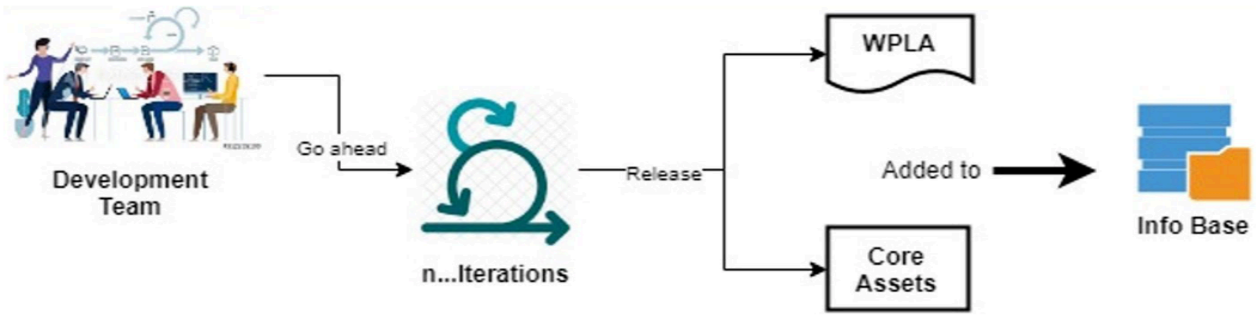


Figure 17 - Agile DE process (Kiani, 2021)

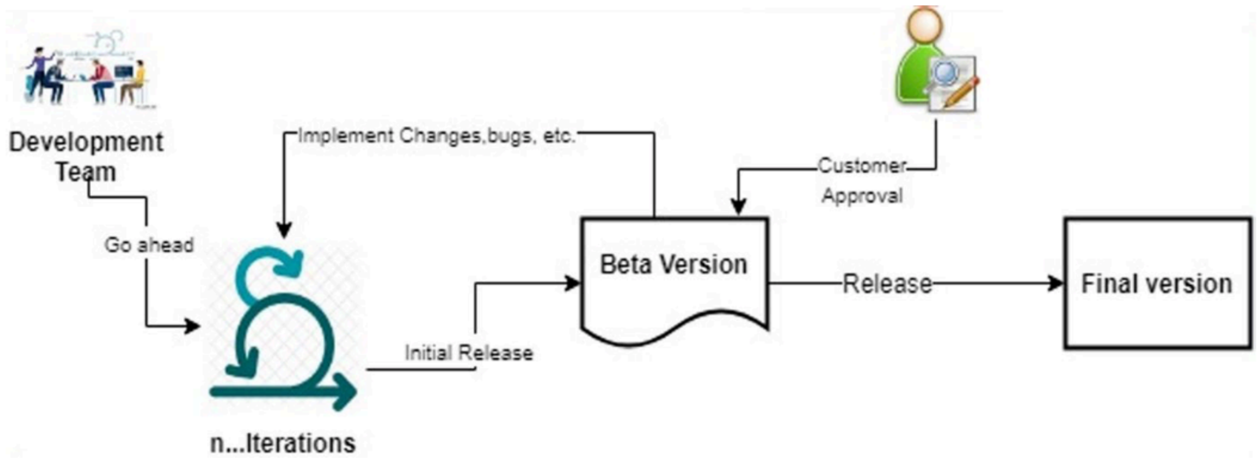


Figure 18 - Agile AE process (Kiani, 2021)

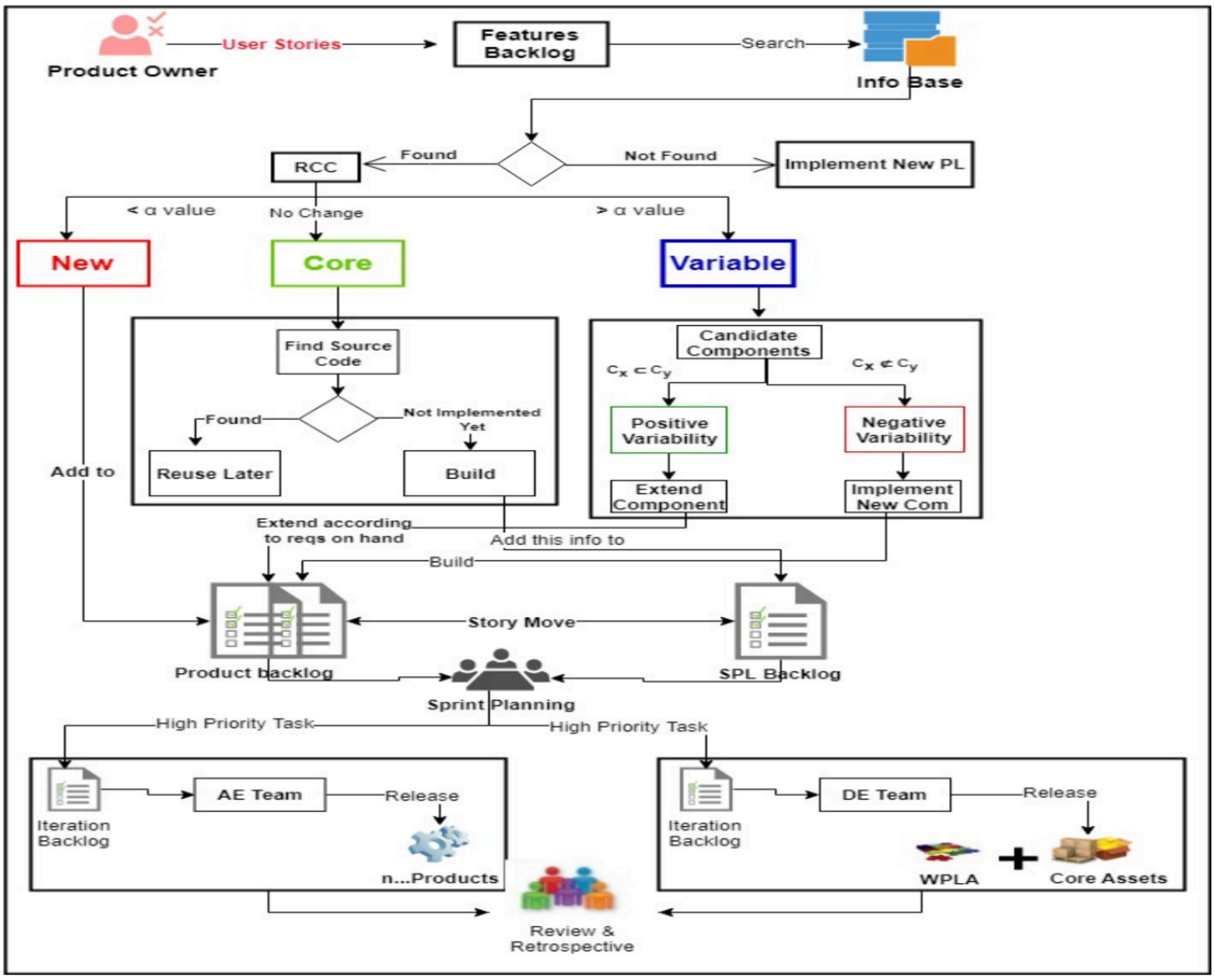


Figure 19 - APLE method proposed by Kiani et al. (2019, 2021)

5.3 Agile SPL Scoping

Da Silva et al. (2012) propose an SPL scoping approach combined with Agile fundamentals or techniques. The first phase presented is the Define pre-scoping that use Onsite interaction session agile practice and collaboration engineering patterns to understand what products or sub-domains consider. After that features are determined and sharpened through Feature Driven Development (FDD) and Feature Oriented Domain Analysis (FODA) with the stakeholders presence. In the commonality and variability analysis it was proposed the use of Pair Programming and Shared Code activities to have a shared vision of the products. Product map and Feature model are generated and Da Silva (2012) suggest to carry out the models examination task simultaneously with the other activities to improve the iterativeness. Furthermore propose the use of Model Storming to represent and comprehend faster the features and their interconnections.

After that come the Release scope phase with the feature prioritization task, based on value, potential change, reuse and risk, the estimate feature and set team velocity tasks to establish the most significant feature. Based on these activities a list of features is issued.

In conclusion the features to be executed are selected with the contribution of the Planning game practice and the Specify acceptance test for leaf feature activity, respectively to evaluate the effort and connect the method to requirements or testing (Da Silva, 2012).

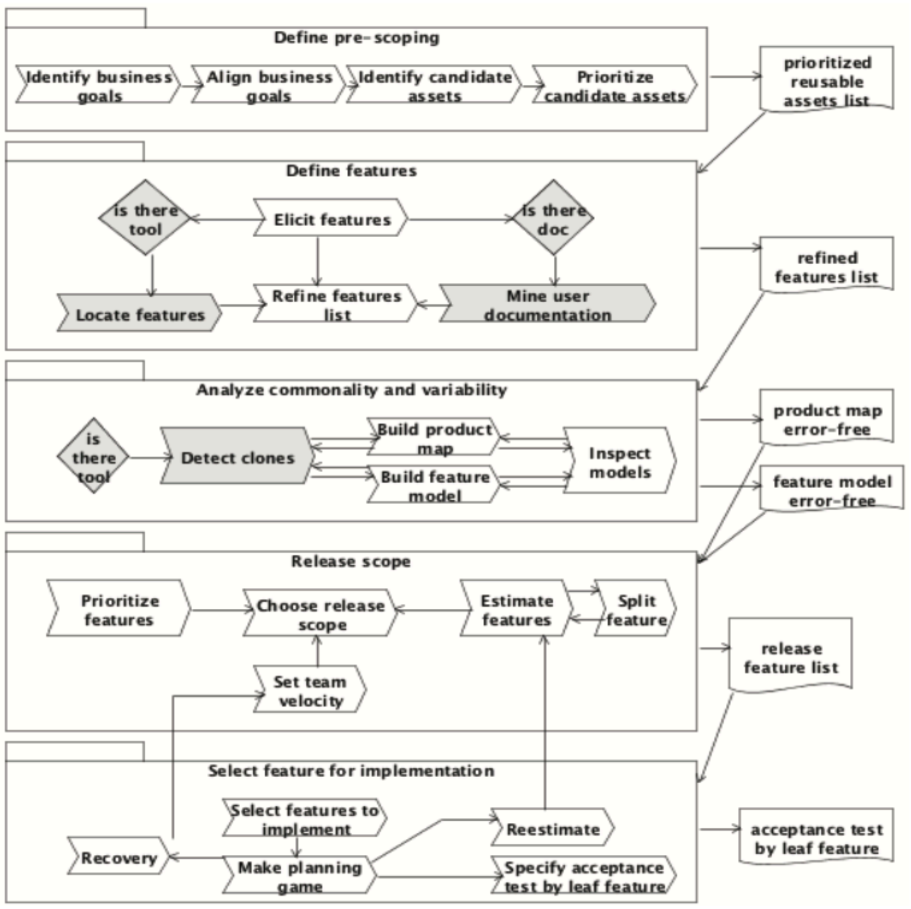
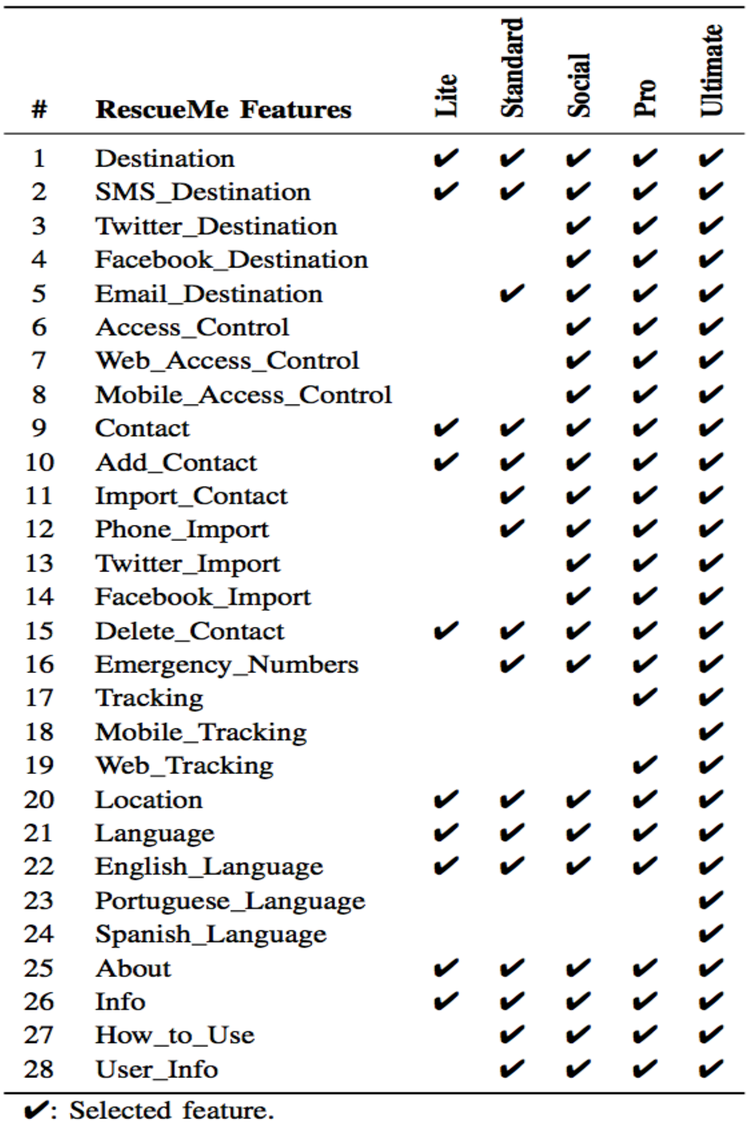


Figure 20 - Agile SPL Scoping (Da Silva et al., 2012)

5.4 SPLICE

Vale et al. (2014) proposed an approach called SPLICE that merge Software product line engineering (SPLE) and Scrum agile methodology. Scope Owner role is presented similar to the Scrum Product Owner. In the Portfolio Planning phase Scope Owner and Product Expert identify products, identify significant features, build a product map and a feature model and prioritize features. The list of organized features is called Scope Backlog.

The Sprint Development phase allows to determine Sub-features, analyze commonality and variability as a result of the feature model and the product map. A case study of a product line mobile applications is shown. Tracking, Contact, Language, User info are some of the features considered an the product map generated is shown in Figure 21.



#	RescueMe Features	Lite	Standard	Social	Pro	Ultimate
1	Destination	✔	✔	✔	✔	✔
2	SMS_Destination	✔	✔	✔	✔	✔
3	Twitter_Destination			✔	✔	✔
4	Facebook_Destination			✔	✔	✔
5	Email_Destination		✔	✔	✔	✔
6	Access_Control			✔	✔	✔
7	Web_Access_Control			✔	✔	✔
8	Mobile_Access_Control			✔	✔	✔
9	Contact	✔	✔	✔	✔	✔
10	Add_Contact	✔	✔	✔	✔	✔
11	Import_Contact		✔	✔	✔	✔
12	Phone_Import		✔	✔	✔	✔
13	Twitter_Import			✔	✔	✔
14	Facebook_Import			✔	✔	✔
15	Delete_Contact	✔	✔	✔	✔	✔
16	Emergency_Numbers		✔	✔	✔	✔
17	Tracking				✔	✔
18	Mobile_Tracking					✔
19	Web_Tracking				✔	✔
20	Location	✔	✔	✔	✔	✔
21	Language	✔	✔	✔	✔	✔
22	English_Language	✔	✔	✔	✔	✔
23	Portuguese_Language					✔
24	Spanish_Language					✔
25	About	✔	✔	✔	✔	✔
26	Info	✔	✔	✔	✔	✔
27	How_to_Use		✔	✔	✔	✔
28	User_Info		✔	✔	✔	✔

✔: Selected feature.

Figure 21 - Product map (Vale et al., 2014)

5.5 ScrumPL

Santos & Lucena (2010) presented a method called ScrumPL that connects the Scrum agile practice with Software Product Line Engineering (SPLE). In particular the approach is divided in Domain Engineering (DE) and Application Engineering (AE) in accordance with the SPLE and consists of Scrum phases as Planning, Staging, Development and Release.

In the DE process product features are identified and attached to the product backlog and reusable features are detected in the AE process.

The product owner, who is also the architect, is accountable for building and conserving the architecture and inserting components into the product backlog.
They show the model in a Tv navigation System case study in which the reference architecture is shown in Variability and Component Diagrams.
Language, Market segment and Standard components are identified as variation points and their sub features, like English language and Portuguese language, are added to the product backlog and classified between Low, Medium, High prioritization value.

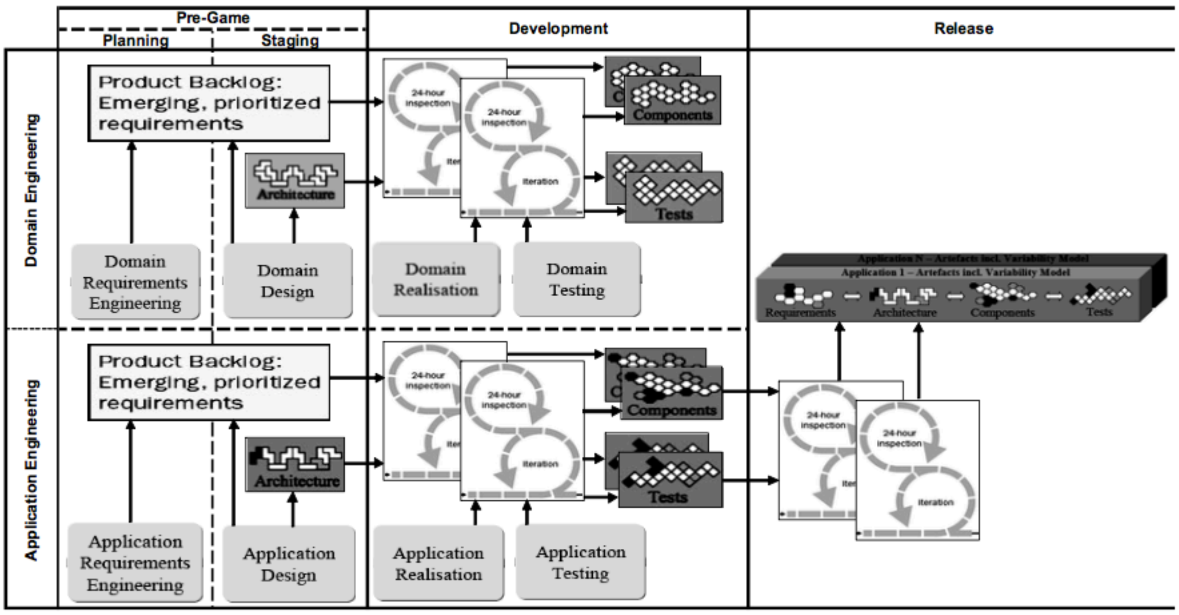


Figure 22 - ScrumPL process Overview (Santos & Lucena, 2010)

5.6 A-Pro-PD

O'Leary et al. (2012) presented an agile product derivation approach called A-Pro-PD.
This method is divided into three phases that are Preparation for Derivation, Product Configuration and Product Development and Testing. The first phase is important for defining product requirements and scoping the product. The second phase aims to build a partial product configuration that meet product requirements through reuse of platform assets. The last phase intends to fulfill requirements not satisfied by reusable platform artefacts.
O'Leary et al. (2012) embrace the "early and continuous delivery of valuable software" agile rule, for this reason the Product Team apply alterations at product level and if this variations could be reusable the Platform Team extracts them from the product. They suggest the use of pair programming methodology to implement and reassess product alterations and planning game technique to control product iterations. This method not include Scrum framework and product backlog.

5.7 Backlog management and Feature model

Raatikainen et al. (2008) propose an approach that employs Agile practices in Software Product family development. In particular Kumbang and Agilefant were considered and shown in Figure 23. Feature model is the key factor of Kumbang and for this reason this method allows to shape the product family at every level. Features are decomposed in sub-features until the leaf nodes are obtained and constraints are identified. On the other side, Agilefant permits to manage backlogs. The Product backlog is constituted by one or more feature backlog items, corresponding to the Kumbang leaf features. Objects in development are merged into iteration backlog and moved to detailed backlog items to be implemented. The feature backlog item is intended finalized when all its items are done.
This combined approach allows to have a whole view of the architecture, functionality and backlogs of the product family (Raatikainen et al., 2008).

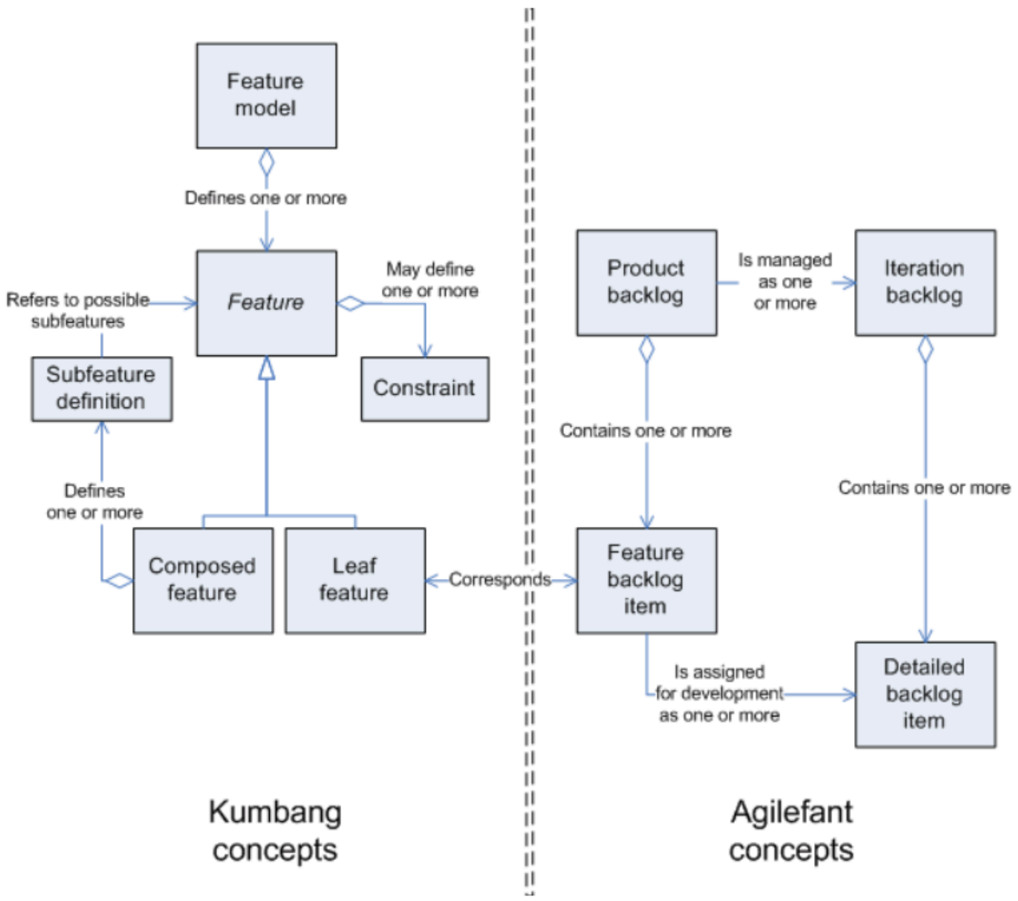


Figure 23 - Kumbang and Agilefant concepts (Raatikainen, 2008)

5.8 APLA

Diaz et al.(2014) presented an in-depth case study of a family of power metering management applications in the domain of Smart Grids to demonstrate that it is possible to combine agile development approaches and methodologies to develop a product-line. To prove this statement the approach called Agile Product-Line Architecting (APLA) is applied. This approach wants to merge the Product-Line Architecture (PLA) model, a method to build a product-line platform, with the Agile Scrum development method, as shown in Figure 24.

Diaz et al. (2014) employ a method called Flexible-PLA that aims to make the architecture more flexible and adaptable throughout its creation. The main principle of this method is the Plastic Partial Component (PPC) which shows the component internal variability by using Variation points.

Points of variability are connected to portions of code, known as variants, that can be used to extend PPCs. The weavings instead indicate how to add these variants.

They use the PLAK metamodel, a tool for documenting knowledge and tracking features during their implementation, which permits to indicate whether a design decision is closed or open and can also show sub decisions and alternatives. The PLAK metamodel makes it possible to obtain a link between

features and architecture solutions, moreover it is lightweight as it only acquires necessary information and after each iteration the knowledge is improved and updated.
Furthermore a change impact analysis was applied to understand what happen if a new feature is introduced. In this way, through the traceability-based algorithm and the propagation rule, connections are formed between the features and the architecture.
The APLA process adjusts the Scrum framework to focus on building the product line architecture in each sprint. For this reason the architect was added to the Scrum Team with the aim of controlling the architecture and its constraints. Scrum was applied in both Software Product Line phases, i.e. Domain Engineering and Application Engineering, respectively to build the common platform and develop individual products. In the beginning the Software Product-Line Owner translate the product vision into features, subsequently decomposed in User Stories and prioritized based on business value. The tasks related to the agile construction of the PLA are developed in the Sprint Planning Meeting, so that it is possible to reorder the features if needed. The APLA process involve three phases. In the first step the architects analyze, through the change impact analysis algorithm, the Working PLA deriving from the previous sprint. In this way they can perceive how the system changes due to the addition of the planned features for the current sprint and they could reprioritize them. In the second step, features are added to the Working PLA and architects implement them using the Flexible-PLA model. In the last step all the decisions, constraints and dependencies are documented through the PLAK model. As output of the sprint planning are obtained: (i) Sprint Backlog, (ii) Change-impact knowledge and (iii) Flexible-PLA and PLAK models that have to be executed in the sprint. The developers implement the Working PLA, that consists of common and variable assets, using as input the Flexible-PLA and PLAK models. Instead, in the Application Engineering, working products are implemented. The Sprint Review and Sprint retrospective meetings are executed and when the Sprint is completed, Increments of Working PLA and Working products are achieved. In the case study were present six developers, two product owner, one Scrum master, who played also the role of part-time architect, and a full time architect.
Features were defined by the Product Owner in a detailed way, for example the first feature was described as: "*F2_Meter storing. It consists of a large data store running over an object-oriented NoSQL database (specifically, Big Data Oracle running over Berkeley DB).*" A feature model that shows the product family structure is shown and the features are broken down into user stories.
For example the feature F2 shown above was decomposed in two user stories: *"(i) installation and configuration of the database manager (Berkeley DB)"* and *"(ii) several conceptual proofs to create and access the database "*. Some missing things were found: The prioritization of the user stories is based on business value, but it is not explain exactly how this activity is executed; User stories are

not defined in the template “As a ⟨role⟩, I want ⟨goal⟩, [so that ⟨benefit⟩]”; The Product backlog artifacts was not shown.
As a result this approach is the only one that has fully applied the Scrum methodology and that shows an extensive case study.

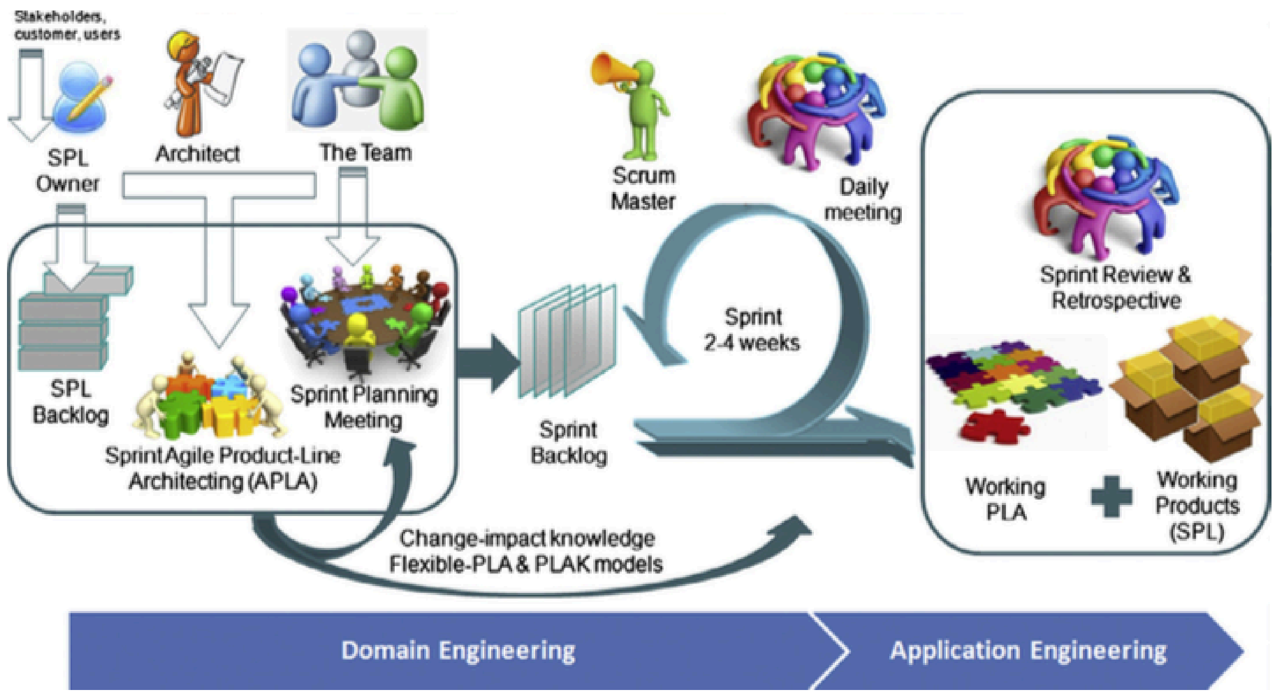


Figure 24 - APLA approach proposed by Diaz (2014)

A list of the main steps of the APLA model is shown in the Table 2.
As a result of this case study, the Flexible-PLA modeling produce flexible and adaptable architectures, Flexible-PLA and PLAK model allow to keep a record of PLA and the change impact analysis detects the impact of the new feature and assists architects taking better decisions.
The change impact analysis consists of traceability-based algorithm that provide a set of design decisions or architectural elements affected by the changes in features, and rule-based inference engine that simulates changes in the working architecture to see the propagation and the effects.

Agile Product-Line Archietecting (APLA) Process (Diaz, 2014)	
REQUIREMENTS	Product owner captures requirements from the Product vision (features) and describes them in detail.
FEATURE MODEL	Features are represented in a hierarchical tree structure
PRODUCT BACKLOG	The product backlog is generated. Features are decomposed in User stories, prioritized based on business value and assigned to sprints.
SPRINTS	Start implementing the feature related to the specific sprint
	User stories planned for the specific Sprint are considered (Sprint backlog)
	Change impact analysis evaluates the repercussions on the Working PLA, due to the insertion of the new feature. As a result architects can re-prioritize the features and define where and how implement them.
	Features are implemented in the Working PLA through the Flexible-PLA model. Architects model the Working PLA adding, deleting or adjusting: - Components - PPCs - Variants - Variability Points - Connections - Optional Connections
	Plak model keeps track of all Components and Design decisions (and the rationale behind).

Table 2 - APLA model steps

Author	Method	Information	Agile Metodology	Backlog type	Sprint	Roles
Raatikainen (2008)	-	Features	Kumbang - Agilefant	Feature Backlog Product Backlog	-	-
Santos & Lucena (2010)	ScrumPL	Features	SCRUM	Product Backlog Sprint Backlog	Sprint Sprint Review Sprint Planning	Product Owner (Also Architect)
Da silva et al. (2012)	Agile SPL Scoping	Features	Feature Driven Development (FDD) Planning game	-	-	-
O' Leary et al. (2012)	A-Pro-PD	Features	Pair programming	-	-	-
Vale et al. (2014)	SPLICE	Features	SCRUM	Scope backlog	Sprint Development Sprint Planning Sprint Review Sprint Retrospective	Scope Owner Product Expert SPL Expert Scrum Master
Haidar et al. (2019)	AgiFPL	Features	Scrumban - SCRUM	Feature backlog (DE) Selected backlog (DE) App backlog (AE) Sprint backlog	Sprint Planning Sprint Execution Sprint Review Sprint Retrospective	Product Owner (App i Owner) Domain Experts (Domain Engineering) Development team
Kiani et al. (2019)	Agile product line egineering (APLE)	Features	SCRUM	Product backlog Feature backlog SPL backlog Sprint backlog	Sprint Sprint Review	Scope Owner SPL Expert SPL Developer SPL Tester
Diaz et al. (2014)	Agile Product Line Architecture (APLA)	Features	SCRUM	Product Backlog Sprint Backlog SPL Backlog	Sprint Planning Sprint Review Sprint Retrospective	SPL Owner Architects Developers

Table 3 – AGILE SPL SCOPING APPROACHES

6 HARDWARE AND LARGE-SCALE AGILE

New studies on the use of Large-scale Agile frameworks are appearing, but despite the growing use of these methodologies, there is still a lack of evidence of their use in the scientific literature (Uludag et al., 2019). Pradhan et al. (2021) declare that Cysco System Inc. will still take several years to complete the Agile transition. Furthermore, companies that are scaling agile are normally mixing Waterfall and Agile development methodologies becoming Hybrid organizations (Pradhan & Nanniyur, 2021). As also reported by Bohmer et al.(2018), despite the wide use of Scrum framework, there are still isolated cases showing its use in hardware field.

According to Bosch (2016) the Software can get faster releases than hardware, for this reason the system architecture is usually separated to allow independence between the two areas. As reported by Berg et al. (2020) hardware startups are able to obtain prototypes quickly through evolutionary methods, hardware-software decomposition approaches and appropriate Agile practices. It has been found a case study of developing a product platform of a family of power transformers but it is not providing a depth explanation and seems like the Scrum agile methodology has been applied only at the team level and not at the product level, then there are not information about the product backlog (Varl et al., 2020).

Žužek et al. (2020) proposed an Agile-Concurrent hybrid framework that wants to combine the Scrum agile method with concurrent product development. This proposed framework maintains the concurrent engineering model as its basic structure introducing collaboration with customers, scope adjustability, team self-organization, Scrum events and roles.

According to Vinodh et al. (2010), the use of CAD and rapid prototyping allows to obtain agility in the development of physical products. Berg et al. (2020) also report the importance of rapid prototyping to receive customer feedbacks, exhibiting the difficulty in the hardware environment, due to long production and shipping times. Simulation is also considered an useful tool to satisfy quality attributes of physical products.

West (2011) introduced another hybrid method called Water-Scrum-Fall, where the Scrum framework is used just in the development process. Instead the Upfront design and Release frequency are more similar to the Waterfall approach.

Another hybrid methodologies for physical new product, called the Agile-stage-gate model, was proposed by Cooper (2016, 2018, 2020) , the creator of the Stage-Gate approach explained in the first Chapter. According to Cooper it’s possible to integrate Scrum into Stage-Gate, not only in the development phases, but also in earlier stages or even in the launch stage. The problem in the physical field is that it’s not possible to build a working product deliverable at the end of a single sprint, as in the software development. For this reason Cooper proposed the use of "protocept", something in the middle between product concept and ready-to-trial prototype. They can be computer-generated 3D drawings, virtual prototypes, crude models, working models, rapid prototypes, or early prototypes, something that is possible to show to obtain the customer feedback. Furthermore, in the product definition, only a part of the requirements should be fixed (40-to-70%), because the product is still unknown.

7 DISCUSSION

Various affinities have been found between product platform design of physical product and Software product line. In the customer needs analysis of tangible product platform and Software product line are used same methods such as Kano model and Quality function deployment (QFD) (Alsawalqah et al., 2014; Cheng et al., 2015). It was proposed to use the Kano model also in the Agile Product management with Scrum by Pichler (2010) .

Another common thing between product platform development and SPL is the use of algorithm and integer linear programming to create the product platform or optimize the scope (Alsawalqah et al., 2014; ElMaraghy & Moussa, 2019)

In the tangible product platform design methods are found similarities with the Feature model. The Product Family Structure Tree (PFST) of Zhang et al. (2019) is comparable to the feature model, but instead of features there are product modules composed by structure, function and process parameters, then a broader view based also on physical and process domain. However this method does not show product variants and options as the feature model.

The Agile artifact Product backlog applied to product platform (i.e. *Product Platform Backlog*) in the field of physical products was not found, but in its place a *Software Product Line (SPL) backlog* was found in software product line scoping (Díaz et al., 2014; Kiani et al., 2019, 2021). The existence of an SPL backlog indicates the possibility that a Product Platform Backlog may exist also in the context of physical products.

As it is possible to derive the product backlog and the SPL backlog from the feature model, we can assume that it is possible to do the same with enhanced function-means (EF-M) tree to obtain a product platform backlog for tangible products.

By analyzing Johannesson's case study of physical product platforms, and Diaz's case study of Agile SPL architecting, some similarities and differences were found. First of all, both methods are focused on customer needs. Both approaches seems Flexible and Adaptable to changes. Johannesson obtained

this properties through the use of the Configurable Component (CC) model and the parametric bandwidth, instead Diaz succeeded employing the Plastic Partial Component (PPC) concept. More specifically, the CC model provides modularity, the parametric bandwidth give scalability and the PPC support both of them. Johannesson uses the Design Rationale (DR) concept to describe the reason why things are the way they are, for example why a certain Design Solution was chosen, and the DR is refined repetitively. Diaz on the other side uses the PLAK model to encapsulate knowledge and design decisions. The global vision of the product family is shown by the CC model in Johannesson and by the Feature model in Diaz. Comparing these two methods it was found that they are similar and they both provides variants information but the CC model consists of physical components. Only in the more detailed EF-M tree Functional Requirements are shown. The EF-M is composed mainly of triplets, i.e. FR, DS and C. These elements are connected by arrows with a small wording such as "isb" which means "is_solved_by". When these acronyms are many, it becomes difficult to understand the type of relationship between the various elements. Instead in the feature model it is easier to understand the connections because they are shown in a simple and graphical way. Johannesson also reports that is important including customers and pre-production engineers in the development process. It seems like, as in the agile development, also in the traditional product development it is more and more present the customer centricity.
Normally one of the problem of physical products is that are difficult to adjust when requirements change. Therefore, they are inflexible to manage during the development phases of the platform. But this problem can be reduced through the concepts of elaboration and encapsulation, which respectively allow to look at the detail and have a global vision of the system (Johannesson et al., 2017).
Various authors have tried to apply agile methodologies to the development of product families, but only a few have provided in-depth case studies to demonstrate their possible implementation. The case study shown by Diaz et al.(2014) is the only one who has managed to fully apply the Scrum methodology to the development of a platform product-line. None of the authors have applied the Scrum methodology exactly as in the Scrum guide, each one made some changes to customize the method. For example, Kiani et al. (2021) split the product backlog in three artifacts, an initial Feature Backlog, SPL Backlog in Domain Engineering and Product backlog in Application Engineering, allowing the movement of stories between SPL backlog and Product backlog. A general view of several differences are shown in Table 2. Normally, in the SPL, the Scoping phase is carried out at the beginning, but with the introduction of agile methodologies, it is spread over the entire development process, as can be seen in Diaz et al. (2014) and Kiani et al. (2019).

Another very common thing is the use of databases to collect artifacts as, for example, shown by Kiani et al. (2019), with the "Info Base", and Haidar et al. (2017), with the "Warehouse".
Furthermore, not all approaches encountered that use agile methodologies employ the product backlog as shown in Da silva et al. (2012) and O'Leary et al. (2012). Moreover, not all those who use the product backlog also use the User stories, for example the product backlog shown by Santos & Lucena (2010) consists of product backlog items, represented by the requirements name, Prioritization, Estimate (size) and Sprint columns.
As shown in chapter 6, analyzing few cases of software-hardware embedded system companies, what emerges is a coexistence between plan-driven (waterfall) and agile methodologies, caused by the long time needed for the agile transformation. In general, there is the tendency to combine Agile and Traditional methodologies in the product development.
Most of the Agile SPL Scoping approaches found are based on Scrum. Large-scaled frameworks such as Scaled Agile Framework (Safe), Large-Scale Scrum (Less) or Disciplined Agile Delivery (Dad) are starting to be wisely used, but there is not a large presence of case studies.
Another hybrid methodologies for physical new product, called the Agile-stage-gate model, was proposed by Cooper (2016, 2018, 2020) , the creator of the Stage-Gate approach explained in the first Chapter. According to Cooper it's possible to integrate Scrum into Stage-Gate, not only in the development phases, but also in earlier stages or even in the launch stage. The problem in the physical field is that it's not possible to build a working product deliverable at the end of a single sprint, as in the software development. For this reason Cooper proposed the use of "protocept", something in the middle between product concept and ready-to-trial prototype. They can be computer-generated 3D drawings, virtual prototypes, crude models, working models, rapid prototypes, or early prototypes, something that is possible to show to obtain the customer feedback. Furthermore, in the product definition, only a part of the requirements should be fixed (40-to-70%), because the product is still unknown.

8 CONCLUSION

This research aimed to find a connection between product platform development and agile development in software and physical product field. In particular, the purpose of the study was to find case studies of the application of Agile frameworks, as Scrum, in the product platform scoping approach. As a result, the possibility of creating agile product platforms in the software field was found in the case study presented by Diaz et al. (2014). It was shown how the agile methodology combined with a flexible product line architecture can provide an early and continuous delivery of valuable software. Furthermore, the possibility of creating software platforms ready to accommodate changes in requirements at any time, even in the final stages of development, was demonstrated.
The research found a lack of literature in the application of Agile frameworks in physical product platform development and in general there is a shortage of empirical studies addressing the employment of large-scale Agile framework.
In addition, similarities between the agile approach of Diaz et al. and Johannesson et al. were found, Furthermore, similarities were found between the methodologies shown by Johannesson in the development of physical product platforms and the methodologies presented by Diaz in the agile software product line development.
Due to the similarities found between the methodologies analyzed, as suggestions for the future, the Scum framework could be applied to the development of physical product platforms, as has been done for software product families.

9 References

Abrantes, R., & Figueiredo, J. (2014). Feature based process framework to manage scope in dynamic NPD portfolios. *International Journal of Project Management*, *32*(5), 874–884. https://doi.org/10.1016/j.ijproman.2013.10.014

Akao, Y., & Mazur, G. H. (2003). The leading edge in QFD: Past, present and future. *International Journal of Quality & Reliability Management*, *20*(1), 20–35. https://doi.org/10.1108/02656710310453791

Alsawalqah, H. I., Kang, S., & Lee, J. (2014). A method to optimize the scope of a software product platform based on end-user features. *Journal of Systems and Software*, *98*, 79–106. https://doi.org/10.1016/j.jss.2014.08.034

Baylis, K., Zhang, G., & McAdams, D. A. (2018). Product family platform selection using a Pareto front of maximum commonality and strategic modularity. *Research in Engineering Design*, *29*(4), 547–563. https://doi.org/10.1007/s00163-018-0288-5

Beck, K., Beedle, M., Cockburn, A., Schwaber, K., & Sutherland, J. (2001). *Manifesto for Agile Software Development*. http://agilemanifesto.org/

Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, *35*(6), 615–636. https://doi.org/10.1016/j.is.2010.01.001

Benington, H. D. (1983). *Production of Large Computer Programs*. *5*(4).

Berg, V., Birkeland, J., Nguyen-Duc, A., Pappas, I. O., & Jaccheri, L. (2020). Achieving agility and quality in product development - an empirical study of hardware startups. *Journal of Systems and Software*, *167*. https://doi.org/10.1016/j.jss.2020.110599

Bohmer, A. I., Hugger, P., & Lindemann, U. (2018). Scrum within hardware development insights of the application of scrum for the development of a passive exoskeleton. *2017 International Conference on Engineering, Technology and Innovation: Engineering, Technology and Innovation Management Beyond 2020: New Challenges, New Approaches, ICE/ITMC 2017 - Proceedings*, *2018-Janua*, 790–798. https://doi.org/10.1109/ICE.2017.8279965

Bosch, J. (2016). Speed, Data and Ecosystems: The Future of Software Engineering. *IEEE Software*.

Boute, R. N., Van Den Broeke, M. M., & Deneire, K. A. (2018). Barco implements platform-based product development in its healthcare division. *Interfaces*, *48*(1), 35–44. https://doi.org/10.1287/inte.2017.0917

Cheng, X., Lan, G., & Zhu, Q. (2015). Scalable product platform design based on design structure matrix and axiomatic design. *International Journal of Product Development*, *20*(2), 91–106. https://doi.org/10.1504/IJPD.2015.068962

Chowdhury, S., Maldonado, V., Tong, W., & Messac, A. (2016). New modular product-platform-planning approach to design macroscale reconfigurable unmanned aerial vehicles. *Journal of Aircraft*, *53*(2), 309–322. https://doi.org/10.2514/1.C033262

Chowdhury, S., Messac, A., & Khire, R. A. (2011). Comprehensive product platform planning (CP3) framework. *Journal of Mechanical Design, Transactions of the ASME*, *133*(10). https://doi.org/10.1115/1.4004969

Claesson, A., & Johannesson, H. (2006). Integrated and configurable product and manufacturing system models. *9th International Design Conference, DESIGN 2006*, 791–798.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley. https://books.google.pt/books?id=SvIwuX4SVigC

Cooper, R. G. (1990). *Stage-Gate Systems: A New Tool for Managing New Products*. *June*.

Cooper, R. G. (2008). Perspective: The stage-gates® idea-to-launch process - Update, what's new, and NexGen systems. *Journal of Product Innovation Management*, *25*(3), 213–232.

https://doi.org/10.1111/j.1540-5885.2008.00296.x

Cooper, R. G., & Sommer, A. F. (2016). The Agile–Stage-Gate Hybrid Model: A Promising New Approach and a New Research Opportunity. *Journal of Product Innovation Management*, *33*(5), 513–526. https://doi.org/10.1111/jpim.12314

Cooper, R. G., & Sommer, A. F. (2018). Agile–Stage-Gate for Manufacturers: Changing the Way New Products Are DevelopedIntegrating Agile project management methods into a Stage-Gate system offers both opportunities and challenges. *Research Technology Management*, *61*(2), 17–26. https://doi.org/10.1080/08956308.2018.1421380

Cooper, R. G., & Sommer, A. F. (2020). New-Product Portfolio Management with Agile: Challenges and Solutions for Manufacturers Using Agile Development Methods. *Research Technology Management*, *63*(1), 29–38. https://doi.org/10.1080/08956308.2020.1686291

Da Silva, I. F. (2012). An agile approach for software product lines scoping. *ACM International Conference Proceeding Series*, *2*, 225–228. https://doi.org/10.1145/2364412.2364450

Díaz, J., Pérez, J., & Garbajosa, J. (2014). Agile product-line architecting in practice: A case study in smart grids. *Information and Software Technology*, *56*(7), 727–748. https://doi.org/10.1016/j.infsof.2014.01.014

Elgh, F., Johansson, J., Stolt, R., Lennartsson, M., Heikkinen, T., & Raudberget, D. (2018). Platform models for agile customization – what's beyond modularization? *Advances in Transdisciplinary Engineering*, *7*, 371–380. https://doi.org/10.3233/978-1-61499-898-3-371

ElMaraghy, H., & Moussa, M. (2019). Optimal platform design and process plan for managing variety using hybrid manufacturing. *CIRP Annals*, *68*(1), 443–446. https://doi.org/10.1016/j.cirp.2019.03.025

Gedell, S., & Johannesson, H. (2013). Design rationale and system description aspects in product platform design: Focusing reuse in the design lifecycle phase. *Concurrent Engineering Research and Applications*, *21*(1), 39–53. https://doi.org/10.1177/1063293X12469216

Haidar, H., Kolp, M., & Wautelet, Y. (2017). Agile product line engineering: The AgiFPL method. *ICSOFT 2017 - Proceedings of the 12th International Conference on Software Technologies*, *Icsoft*, 275–285. https://doi.org/10.5220/0006423902750285

Haidar, H., Kolp, M., & Wautelet, Y. (2019). Formalizing agile software product lines with a RE metamodel. *ICSOFT 2018 - Proceedings of the 13th International Conference on Software Technologies*, *Icsoft*, 90–101. https://doi.org/10.5220/0006849000900101

Hinterreiter, D., Linsbauer, L., Feichtinger, K., Prähofer, H., & Grünbacher, P. (2020). Supporting feature-oriented evolution in industrial automation product lines. *Concurrent Engineering Research and Applications*. https://doi.org/10.1177/1063293X20958930

Johannesson, H., Landahl, J., Levandowski, C., & Raudberget, D. (2017). Development of product platforms: Theory and methodology. *Concurrent Engineering Research and Applications*, *25*(3), 195–211. https://doi.org/10.1177/1063293X17709866

Kamrani, A. K., Salhieh, S. E. M., & Salhieh, S. M. (2002). *Product Design for Modularity*. Springer US. https://books.google.st/books?id=m0xEhCPRC0oC

Kang, K. C. (1990). *Feature-oriented Domain Analysis (FODA): Feasibility Study ; Technical Report CMU/SEI-90-TR-21 - ESD-90-TR-222. November*. https://books.google.es/books?id=yYi5PgAACAAJ

Kano, N., Seraku, N., Takahashi, F., & Ichi Tsuji, S. (1984). *Attractive Quality and Must-Be Quality*.

Kantorovich, L. V. (1939). Mathematical Methods of Organizing and Planning Production. *Management Science*. https://doi.org/10.1287/mnsc.6.4.366

Kiani, A. A., Hafeez, Y., Anwar, N., & Abbas, G. (2019). A new approach for agile product line engineering. *Proceedings - 22nd International Multitopic Conference, INMIC 2019*, 1–7. https://doi.org/10.1109/INMIC48123.2019.9022798

Kiani, A. A., Hafeez, Y., Imran, M., & Ali, S. (2021). A dynamic variability management approach working with agile product line engineering practices for reusing features. In *Journal of Supercomputing* (Issue 0123456789). Springer US. https://doi.org/10.1007/s11227-021-03627-

Kim, K. J., Lee, D. U., & Lee, M. S. (2006). Determining product platform elements for mass customisation. *International Journal of Productivity and Quality Management*, *1*(1–2), 168–182. https://doi.org/10.1504/ijpqm.2006.008379

Klünder, J., Hohl, P., & Schneider, K. (2018). *Becoming Agile while preserving software product lines*. 1–10. https://doi.org/10.1145/3202710.3203146

Landahl, J., Jiao, R. J., Madrid, J., Söderberg, R., & Johannesson, H. (2020). Dynamic platform modeling for concurrent product-production reconfiguration. *Concurrent Engineering Research and Applications*. https://doi.org/10.1177/1063293X20958938

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley. https://books.google.pt/books?id=Vn44mQEACAAJ

Levandowski, C. E., Jiao, J. R., & Johannesson, H. (2015). A two-stage model of adaptable product platform for engineering-to-order configuration design. *Journal of Engineering Design*, *26*(7–9), 220–235. https://doi.org/10.1080/09544828.2015.1021305

Levandowski, C., Michaelis, M. T., & Johannesson, H. (2014). Set-based development using an integrated product and manufacturing system platform. *Concurrent Engineering Research and Applications*, *22*(3), 234–252. https://doi.org/10.1177/1063293X14537654

Li, Z., Pehlken, A., Qian, H., & Hong, Z. (2016). A systematic adaptable platform architecture design methodology for early product development. *Journal of Engineering Design*, *27*(1–3), 93–117. https://doi.org/10.1080/09544828.2015.1112366

Martin, M. V., & Ishii, K. (2002). Design for variety: Developing standardized and modularized product platform architectures. *Research in Engineering Design*, *13*(4), 213–235. https://doi.org/10.1007/s00163-002-0020-2

McGrath, M. E. (1995). *Product Strategy for High-technology Companies: How to Achieve Growth, Competitive Advantage, and Increased Profits*. Irwin Professional Pub. https://books.google.pt/books?id=qQx2QgAACAAJ

Meyer, M. H., & Lehnerd, A. P. (1997). *The Power of Product Platforms*. Free Press. https://books.google.pt/books?id=PKJuQjSaHp0C

Mizuno, S., & Akao, Y. (1978). *Quality Function Deployment: A Company-wide Quality Approach*. JUSE Press.

O'Leary, P., McCaffery, F., Thiel, S., & Richardson, I. (2012). An Agile process model for product derivation in software product line engineering. *Journal of Software: Evolution and Process*, *24*(5), 561–571. https://doi.org/10.1002/smr.498

Ojeda, M. C. C., Alegría, J. A. H., Rodriguez, F. J. Á., & Melenje, P. H. R. (2018). A Collaborative Method for a Tangible Software Product Line Scoping. *2018 ICAI Workshops, ICAIW 2018 - Joint Proceedings of the Workshop on Data Engineering and Analytics, WDEA 2018, Workshop on Smart Sustainable Cities, WSSC 2018, Workshop on Intelligent Transportation Systems, WITS 2018 and Workshop on Empirical Experien*. https://doi.org/10.1109/ICAIW.2018.8554999

Oliinyk, O., Petersen, K., Schoelzke, M., Becker, M., & Schneickert, S. (2017). Structuring automotive product lines and feature models: an exploratory study at Opel. *Requirements Engineering*, *22*(1), 105–135. https://doi.org/10.1007/s00766-015-0237-z

Pichler, R. (2010). Agile Product Management with Scrum. In *Evolution*. Addison-Wesley.

Pohl, K., Böckle, G., & Van Der Linden, F. (2005). *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer Berlin Heidelberg.

Pradhan, S., & Nanniyur, V. (2021). Large scale quality transformation in hybrid development organizations – A case study. *Journal of Systems and Software*, *171*. https://doi.org/10.1016/j.jss.2020.110836

Prasad, B. (1999). Enabling principles of concurrency and simultaneity in concurrent engineering. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, *13*(3), 185–204. https://doi.org/10.1017/S0890060499133055

Raatikainen, M., Rautiainen, K., Myllärniemi, V., & Männistö, T. (2008). *Integrating product family modeling with development management in agile Integrating Product Family Modeling with Development Management in Agile Methods. January*. https://doi.org/10.1145/1370720.1370728

Raudberget, D. (2010). Practical applications of set-based concurrent engineering in industry. *Strojniski Vestnik/Journal of Mechanical Engineering*, *56*(11), 685–695. https://doi.org/10.5545/149_DOI_not_assigned

Raudberget, D., Elgh, F., Stolt, R., Johansson, J., & Lennartsson, M. (2019). Developing agile platform assets – Exploring ways to reach beyond modularisation at five product development companies. *International Journal of Agile Systems and Management*, *12*(4), 311–331. https://doi.org/10.1504/IJASM.2019.104588

Robertson, D., & Ulrich, K. (1998). Planning for Product Platforms. *Sloan Management Review*, *39*(4), 19–31.

Royce, W. W. (1970). *Managing the Development of Large Software Systems*. https://doi.org/10.7551/mitpress/12274.003.0035

Santos, A., & Lucena, V. (2010). ScrumPL: Software product line engineering with Scrum. *ENASE 2010 - Proceedings of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering*, *July 2010*, 239–244. https://doi.org/10.5220/0003038302390244

Schachinger, P., & Johannesson, H. L. (2000). Computer modelling of design specifications. *Journal of Engineering Design*, *11*(4), 317–329. https://doi.org/10.1080/0954482001000935

Schmid, K. (2002). A comprehensive product line scoping approach and its validation. *Proceedings - International Conference on Software Engineering*, 593–603. https://doi.org/10.1145/581339.581415

Schwaber, K. (2004). *Agile Project Management with Scrum*. Microsoft Press.

Schwaber, K., & Sutherland, J. (2020). *Scrum Guide V7. November*, 133–152.

Simpson, T. W., Bobuk, A., Slingerland, L. A., Brennan, S., Logan, D., & Reichard, K. (2012). From user requirements to commonality specifications: An integrated approach to product family design. *Research in Engineering Design*, *23*(2), 141–153. https://doi.org/10.1007/s00163-011-0119-4

Sobek, D. K., Ward, A. C., & Liker, J. K. (1999). Toyota ' s Principles of Set-Based Concurrent Engineering Toyota ' s Principles of Set-Based Concurrent Engineering Durward K Sobek II. *Sloan Management Review*, *40*(2), 67–83.

Steward, D. V. (1981). Design Structure System: a Method for Managing the Design of Complex Systems. *IEEE Transactions on Engineering Management*, *EM-28*(3), 71–74. https://doi.org/10.1109/TEM.1981.6448589

Suh, N. P., & Sekimoto, S. (1990). Design of Thinking Design Machine. *CIRP Annals - Manufacturing Technology*, *39*(1), 145–148. https://doi.org/10.1016/S0007-8506(07)61022-1

Tjalve, E. (1979). Creation of a Product. *A Short Course in Industrial Design*, 1–15. https://doi.org/10.1016/b978-0-408-00388-9.50004-5

Uludag, O., Kleehaus, M., Dreymann, N., Kabelin, C., & Matthes, F. (2019). Investigating the Adoption and Application of Large-Scale Scrum at a German Automobile Manufacturer. *Proceedings - 2019 ACM/IEEE 14th International Conference on Global Software Engineering, ICGSE 2019*, 22–29. https://doi.org/10.1109/ICGSE.2019.00019

Vale, T., Cabral, B., Alvim, L., Soares, L., Santos, A., Machado, I., Souza, I., Freitas, I., & Almeida, E. (2014). SPLICE: A lightweight software product line development process for small and medium size projects. *Proceedings - 2014 8th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2014*, *September*, 42–52. https://doi.org/10.1109/SBCARS.2014.11

Van den Broeke, M., Boute, R., Cardoen, B., & Samii, B. (2017). An efficient solution method to design the cost-minimizing platform portfolio. *European Journal of Operational Research*, *259*(1), 236–250. https://doi.org/10.1016/j.ejor.2016.10.003

Varl, M., Duhovnik, J., & Tavčar, J. (2020). Agile product development process transformation to support advanced one-of-a-kind manufacturing. *International Journal of Computer Integrated Manufacturing*, *33*(6), 590–608. https://doi.org/10.1080/0951192X.2020.1775301

Vinodh, S., Devadasan, S. R., Maheshkumar, S., Aravindakshan, M., Arumugam, M., & Balakrishnan, K. (2010). Agile product development through CAD and rapid prototyping technologies: An examination in a traditional pump-manufacturing company. *International Journal of Advanced Manufacturing Technology*, *46*(5–8), 663–679. https://doi.org/10.1007/s00170-009-2142-4

West, D. (2011). Water-Scrum-Fall Is the Reality of Agile for Most. *For Application Development & Delivery Professionals*, 2011–2012. http://www.storycology.com/uploads/1/1/4/9/11495720/water-scrum-fall.pdf

Wu, M., & Wang, L. (2012). A continuous fuzzy Kano's model for customer requirements analysis in product development. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *226*(3), 535–546. https://doi.org/10.1177/0954405411414998

Zhang, Q., Peng, W., Lei, J., Dou, J., Hu, X., & Jiang, R. (2019). A method for product platform planning based on pruning analysis and attribute matching. *Journal of Intelligent Manufacturing*, *30*(3), 1069–1083. https://doi.org/10.1007/s10845-017-1305-7

Žužek, T., Kušar, J., Rihar, L., & Berlec, T. (2020). Agile-Concurrent hybrid: A framework for concurrent product development using Scrum. *Concurrent Engineering Research and Applications*, *28*(4), 255–264. https://doi.org/10.1177/1063293X20958541