Evaluating the benefits of a computer-aided software engineering tool to develop and document product configuration systems

Shafiee, Sara; Wautelet, Yves; Friis, Steffan Callesen; Lis, Lukasz; Harlou, Ulf; Hvam, Lars

Published in:
Computers in Industry

Link to article, DOI:
10.1016/j.compind.2021.103432

Publication date:
2021

Document Version
Peer reviewed version

Citation (APA):
Shafiee, S., Wautelet, Y., Friis, S. C., Lis, L., Harlou, U., & Hvam, L. (2021). Evaluating the benefits of a computer-aided software engineering tool to develop and document product configuration systems. *Computers in Industry*, *128*, Article 103432. https://doi.org/10.1016/j.compind.2021.103432

Evaluating the Benefits of a Computer-Aided Software Engineering Tool to Develop and Document Product Configuration Systems

Abstract

Computer-Aided Software Engineering (CASE) tools are popular software programs to support the members of the development team (including analysts, designers, coders, database administrators, and project managers) in building new software systems. Up-to-date and consistent knowledge representation and documentation is crucial for companies developing Product Configuration Systems (PCSs). The literature reports various challenges in PCS development, such as maintenance, documentation, knowledge management, resource and time management, system quality, and communication with domain experts as particularly problematic. A CASE tool tailored to the specific needs of PCS development can prove to be useful in tackling at least some of these challenges. Such a CASE tool has to support product models, which means it has to not only allow the representation of the product core architecture and the optional selectable features, but also ensure consistency between representations (views) and deliver forward or reverse engineering. This enables support and automates, at least partially, the development in general and the implementation stage. The focus and main contribution of this paper is twofold. First, we describe the view-based approach required to fully conceptualise the knowledge to generate PCS software from the CASE tool. To this end, the tool indeed includes four different views to build or edit all the required knowledge. Second, we validate this CASE tool within two case companies, wherein we evaluate its application on a project each time it is used. The results show that the use of the CASE tool increases the quality of PCS documentation and saves time and resources while also improving the PCS's overall quality.

Keywords: product configuration system; computer-aided software engineering (CASE); modelling; development; maintenance; database management system

1. Introduction

Product Configuration Systems (PCSs) are software systems supporting the configuration of a product family. A car is an example of such a product. Today, cars are seldom sold as a fully predefined product but are rather configured following customer requirements. To be engineered in such a way, the car encompasses a significant complexity due to a high number of attributes and constraints. To develop PCS software for a car model's configuration, the knowledge must first be represented, then verified and confirmed by domain experts such as mechanical engineers, electronic engineers, marketing section, designers, etc. Thereafter, this knowledge needs to be coded inside the PCS. Even after deployment, there is a constant need to maintain the knowledge consistent within the PCS software due to the frequent update of the product (e.g. a car model that gets new options, engine types, promo packages, etc.). The development of PCS software is thus an elaborate process that is hard to support and maintain with basic representations made on regular sheets of papers.

PCSs have entered a new stage of maturity and recently received increased attention from both researchers and practitioners. They can support the most important decisions regarding product features and cost [1]. Widely used in various industries, PCSs enable companies to propose alternatives to facilitate their sales and production processes [2–4]. PCSs have many advantages [5], such as a shorter lead time [5–7], fewer errors [8], an increased ability to meet customer requirements regarding product functionality [9], the use of fewer resources [3], optimised product designs [7,10,11], less routine work, improved on-time delivery [12–14], enhancing human-centric application and value for customers and society [15–18]. However, companies face various challenges during different phases of PCS projects. The reported PCS challenges are investigated and categorised in the literature [19,20]. More specifically, these challenges are listed as resource management [21–24], product complexity [8,9,21,25], IT challenges [8,9,26–28], knowledge acquisition challenges [8,29,30], and organisational challenges [8,27,29]. Computer-Aided Software Engineering (CASE) tools are used for developing high-quality, defect-free, and maintainable software and are often associated with methods for the development of information systems together with automated tools that can be used in a software development process [31,32]. CASE tools ensure a check-pointed and disciplined approach and help designers, developers, testers, managers, and other roles to cross the project milestones during development. Moreover, they aim to address the difficulties of developing high-quality, complex software on time and within the budget [33].

Past research in the field of PCS calls for specific solutions and tools for PCS development to solve challenges such as documentation, knowledge acquisition, resource constraints, and product modelling; there is still a critical need for a supporting solution. A CASE tool can support the creation of artefacts and the follow-up of methods found in PCS literature. The whole process of building, validating, and coding complex product knowledge normally takes significant time and effort. In addition, based on the literature, the main investments of PCS projects are software and resource expenses [34–36]. Hence, being able to generate code from the CASE tool for PCS software implementation and retrieve conceptual models from the existing code would deliver high value. This not only automates part of the implementation process, but also allows editing, updating, and transforming the knowledge contained in PCSs at an ex-post stage. This study introduces a framework and CASE tool to solve the mentioned challenges of the development, implementation, communication, and maintenance of the PCS. The research focus is to develop a CASE tool that can provide support for the PCS development life cycle using structured product modelling. The development process built in the CASE-Tool indeed follows the philosophy of model-driven (software) engineering (see [37] [38] for a description of the practice and [39] for an application in an AI context). To this end it combines the support of PCS-specific representations with a transformation engine for a full life cycle support. The CASE tool automates tedious system design and implementation activities and allows the generation of graphical interfaces.

A fully usable PCS is a software system aimed at supporting the configuration of a product destined to a customer and member of a product line. In other words, the architecture of all the products that can be configured and built using the PCS is similar, but different features can be selected for customisation (this is referred to as variability). PCSs are thus in charge of supporting the configuration of a product, itself a member of a line. As far as the CASE tool presented in this paper is concerned, we only tackle the building of PCS for "traditional"[1] products and services (also sometimes services are options in product configurations). Indeed, a lot of work has been done around Software Product Lines (SPL) where the product is not a physical one but (off the shelf) software and the variability concerns the features/functions of software delivered to a client (e.g. an enterprise resource planning system); many CASE tools have been developed for the specifics of SPLs [40]. Software products are not the first target of the artefacts and method supported by the PCS built using our CASE tool. Studying the development of SPL supported by a PCS built with our CASE tool could nevertheless be done in future research. To the best of our knowledge, the CASE tool presented here is the only one for product development support and is thus outside of the SPL field. We pinpoint its main originality characteristic as being (model-)driven by the Product Variant Master (PVM) and Class Responsibility Collaboration (CRC) cards following the method described in previous literature [1,41].

The research presented in this paper takes root in the design science (DS) paradigm [42]; the paradigm aims to deliver generic solutions to known or unknown problems. Hevner et al. [43] demonstrates that the DS approach can be used when creating and evaluating IT artefacts intended to solve identified organisational problems. A DS research strategy aims to develop knowledge that can be used in a specific and direct way to design and implement actions, processes, or systems to achieve desired outcomes in practice [44], which has also been proven to fit the PCS field [45]. The DS approach is supported by an identified need for formal PCS models and inference tools for providing systematic and comprehensive solutions to practitioners [46,47]. This knowledge is developed by engaging with real-life operation management problems or opportunities [44]. The result[44] of a DS research project can be a solution to a problem in terms of an artefact, terminology, etc., and it is here in the form of an engineering tool that can be used by practitioners willing to develop a PCS. For validation, two PCS projects from two selected engineering manufacturing companies were selected and studied through empirical investigations. Both companies reported difficulties, before using the CASE tool, with the development and maintenance of their PCSs, as well as a lack of documentation to support communication with domain experts.

We consequently explicitly formulate the following research question (RQ).

RQ: *What are the concrete benefits of using a CASE tool to support (and partially automate) the model-driven development of PCS software to configure physical products?*

In order to answer this research question, a set of steps have been performed and are summarized here:

1. We have determined the classical features found in CASE-tools by the help of a literature review;
2. We have determined how the PCS software development approach from [1] that is based on the PVM and CRC cards for PCS analysis and the Unified Modelling Language [27,48] can be supported by a CASE-tool at design level. We refer to this as a framework that goes beyond the initial work of [1] because the proposed framework provides a broader development environment and traceability between models to ensure consistency. The CASE tool has been fully developed;
3. The set of generic features has been applied on the produced CASE-Tool;

[1] By "traditional", we mean "non-software".

4. A few case studies have been selected to apply the CASE-tool on. The latter tool being developed by a consultancy company, some of its clients have been selected for this purpose. The application of the CASE tool on the case studies has been studied on the basis of the following factors:
    a. The usability meaning the time required to learn using the tool, a comparison between the time spent on the development of a PCS with and without the case tool and the user satisfaction with respect to it;
    b. A cost-benefit analysis of the use of the CASE tool on the two selected cases;
    c. The feedback of the employees having worked with the CASE tool on its benefits.

The originality of the research lies in the support of PCS software development using the CASE tool aligned with the artefacts for PCS development outlined by Hvam et al. [1]. The latter indeed offers a model-driven framework for the development of PCS. More precisely, Hvam et al. [1] use the PVM and CRC cards as main artefacts for driving software development. As far as the development life cycle is concerned, these artefacts have been incorporated in the Rational Unified Process [49,50] and later in Scrum as the main processes for conducting the software development [36,41]. The CASE tool supports building and editing these artefacts but is, as such, independent of the life cycle used for the development (i.e. plan-driven or agile) so it can be used in a wide variety of contexts. The main contributions of the paper are:

- A framework to build and link the different aspects of PCS software through complementary views. Within these views, the PVM and CRC cards and other complementary sources of knowledge are conceptualised; the CASE tool structures itself around these views to furnish efficient support of development. The latter indeed enables us to build and edit all the knowledge represented in these views. The views are depicted in the paper and illustrated in an example.
- The validation of the CASE tool through two case studies of PCS software development, including case studies presentation, usability test, compliance matrix, and cost benefits analysis.

This paper is organised as follows: Section 2 discusses the relevant literature, while Section 3 elaborates on the method for the research, explaining the key phases of the research, the preliminary work, the view-based framework supported by the CASE tool, and its validation. Section 4 details the views and features of the CASE tool. Section 5 presents the results of the case studies and discusses them, examines the usability of the CASE tool, highlights the potential of the proposed CASE tool through the compliance matrix, and analyses the cost and benefits to evaluate the financial benefits of the CASE tool. Section 6 discusses the threats to validity. Finally, Sections 7, 8, and 9 discusses the limitations, related works and conclusions of the research.

2. Background

2.1. Modelling of product knowledge

Product modelling is a method of representing the structure and knowledge of a product on a relatively visual, abstract level to ensure it is understandable to all the persons concerned. The major elements of a configuration task (problem) are configuration models specifying the set of possible configurations (solutions) and a configuration model together with a defined set of (customer) requirements [46]. Four basic representations of product modelling for PCSs are shown in Figure 1 [51]. The real world is the product knowledge available at a company. The product model represents product knowledge in a structured way, whereas the information model is a formal technical representation of a product model, which is usually based on the unified modelling language

(UML) notation [27,48]. As a technical model often cannot be understood by most domain experts, the product (or phenomenon) model enables domain experts and configuration engineers to communicate in a common language and thus facilitates future updating and documentation changes [1,28].

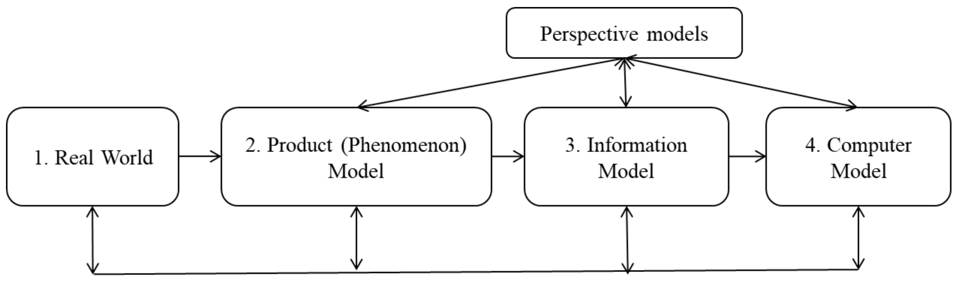


Figure 1. Four basic representations of product modelling for product configuration systems based on the design modelling research approach (revised from [28,51]).

Modelling techniques are used for PCS communication and documentation. Product modelling is used to handle the growing complexities of software development, thereby enabling engineers to work and communicate at higher levels of abstraction and have proper documentation of both the product and project knowledge [52,53]. In addition, these techniques increase the ability to share knowledge between units, which can significantly contribute to an organisation's performance [54]. Different modelling techniques are available for PCSs [52,55]. UML is a general-purpose visual modelling language designed for the specification, visualisation, construction, and documentation of the artefacts of a software system [48]. It encourages designers to formalise their implicit knowledge and makes knowledge extraction easier. For PCS projects, a detailed analysis of the product range is necessary; the use of a PVM together with a CRC card is suggested for this purpose (Fig. 2) – both based on UML techniques. The PVM presented by Hvam et al. [1] displays product knowledge in a structured format, focusing on three different aspects: (1) customer view, (2) engineering view, and (3) production or part view. The use of CRC cards was first put forward by Beck and Cunningham [56]. Hvam et al. [1] have since proposed several revisions to the use of CRC cards in PCS projects, where the classes used are described in further detail.

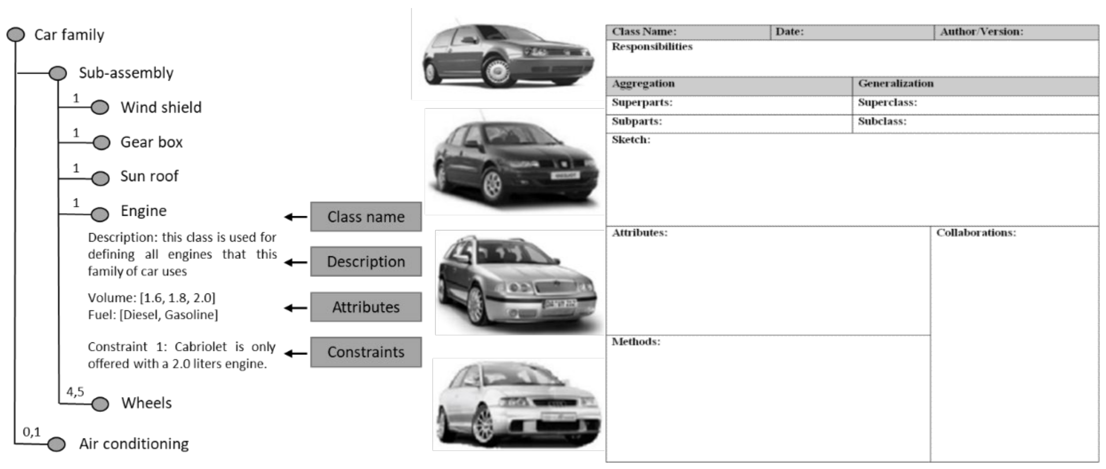


Figure 2. The structure of a product variant master with a car example (left) and class responsibility collaboration (CRC) cards (right) (after [1]).

2.2. Product modelling in commercial configuration systems

This section discusses the available literature regarding commercial PCSs and their structures and levels of maturity to evaluate the possibility of transferring knowledge from one IT system to the PCS. The literature discusses many commercial PCSs that are currently available, such as Tacton, Encoway, and SAP [2]. There are benefits, strengths, and weaknesses of each of the available commercial platforms. Friedrich et al. [2] described modern platforms that must provide a mechanism to abstract the underlying technical representations inside the PCS as much as possible during the modelling phase. The typical elements found in such tools include compatibility tables (which non-IT specialists can understand), a user-oriented rule language (including appropriate editing support), a graphical representation of bills-of-material structures, and testing and tracing support [2].

Several researchers suggested optimal structures for commercial PCSs and listed the requirements. For example, Tiihonen et al. [57] evaluated the modelling efficiency and performance of PCSs in the views of several domain experts and presented a list of requirements for a future, web-based commercial PCS. Moreover, as mentioned previously, researchers agreed that available commercial PCSs should be capable of communicating with non-IT specialists through visualised modelling structures, but modern PCS platforms remain underdeveloped in this regard. The existing literature indicates that modern PCSs do not include sufficient documentation and communication abilities to interact with non-IT experts, or if companies tend to provide accessibility to everyone, the licenses are also expensive. On the contrary, it takes months for the development team to gather and model all the product data inside PCSs and validate them iteratively.

For this study, commercial PCSs available on the market were evaluated. The model structures within these commercial PCSs correspond to the PVM or CRC card approach and contain all the necessary information. More specifically, all the components and sub-components—including the attributes, constraints, and rules—form a tree structure. The goal is to transfer and translate all the relevant knowledge into the documentation system and thus to automatically generate PVMs and CRC cards.

The tree structure present in the PVM describes the product structure, attributes, and constraints, whereas the CRC cards are used to describe individual classes in further detail, as shown in Figure 3. The common drawbacks of product knowledge models in commercial PCS modelling environments are the lack of additional explanation regarding rules or variants and product hierarchy [19]. Furthermore, most of the knowledge is stored in the IT language. These drawbacks indicate that commercial PCSs seem to fall short in terms of the functionality required to document product models and communicate with domain experts. The development tasks also require expert resources. However, the task of PCS modelling can be accomplished automatically if the needed data can be gathered and documented in a CASE tool.

2.3. Documentation for PCSs

One of the main challenges when using PCSs is the difficulties in documentation and maintenance, which can lead to incomplete and outdated systems that are difficult to understand [19,57]. Documentation is a vital part of all IT projects, as it is used for sharing knowledge between people and reducing knowledge loss when team members become inaccessible. For a company using a PCS, it is crucial to have an efficient system for documenting the structure, attributes, and constraints modelled within the system, as well as to facilitate communication between PCS developers and domain experts [20]. Elgh [58] proposed a framework consisting of an information model and the underlying principles to be used when developing a design automation system for quotation preparation. The existing research on configuration documentation indicates that additional specifications were introduced in 2007 for a PCS IT documentation system called product model manager—an environment that provides a simple overview of product elements in the form of CRC cards and PVMs [29]. Shafiee et al. [28] proposed an automatic agile solution to extract knowledge from PCSs, which is also used in this research. However, the proposed solutions, methods, and tools for documenting PCSs, including the automatic solution proposed by Shafiee et al. [28], consider the documentation, updating, and stakeholder involvement focusing only on PCS documentation challenge. Hence, there is a need for a CASE tool to automate and manage all PCS challenges, including data collection, communication, development, documentation, and integration.

2.4. Typical features of CASE tools

Multiple CASE tools have been developed to support various kinds of software development. A CASE tool can be defined as a set of tools and methods that generate a software system to support the desired end result (i.e. a defect-free and maintainable software product) [31]. Research results of CASE tools have focused on four particular aspects [32]: (1) modelling and representation of engineering information; (2) conceptual structure of engineering information; (3) algorithms for transforming different representations of engineering information into semantic conceptual structures; and (4) innovative applications. This research will cover all four aspects. The benefits and typical features associated with CASE tool literature are provided in Table 1. We searched databases such as Google Scholar, Scopus, IEEE Xplore, and Science Direct Elsevier. Through this semi-structure review, we identified a number of alternate keywords represented as: (CASE OR computer-aided software) AND (requirements OR system development life cycle OR database management system OR database administrator OR software product line).

Table 1. Typical features found in CASE tools

	Feature	**Comments**
1	Document handling [59]	The most obvious area for tool support is document handling. Traditional inspection requires the distribution of multiple copies of each document required. Apart from the cost and environmental factors associated with such large amounts of paper, cross-referencing from one document to another can be difficult.
2	Individual preparation [59]	There are various levels of integration, from simply reporting defects to producing an annotation related to the defect for the reviewer to examine.
3	Data collection [59]	Computer support allows metrics from the inspection to be automatically gathered for analysis. This removes the burden of these dull, but necessary, tasks from inspectors themselves, allowing them to concentrate on the real work of finding defects.
4	Automation of the SDLC [60]	CASE tools are usually classified according to the extent of support they provide for the SDLC. For example, front-end CASE tools provide support for the planning, analysis, and design phases; back-end CASE tools provide support for the coding and implementation phases.
5	Easier maintenance of application systems [60]	The documentation and ease of access for them make the maintenance more efficient. Better communication and an overview for experts and project members are provided.
6	Standardisation of system development methodologies (automatic performance) [60,61]	One of the most important components of CASE tools is an extensive data dictionary, which keeps track of all objects created by the system designer.

7	Data dictionary [60]	The CASE data dictionary stores data flow diagrams, structure charts, descriptions of all external and internal entities, data stores, data items, report formats, and screen formats. It also describes the relationships among the components of the system.
8	Interaction with DBMS [60]	Interfaces allow the CASE tool to store its data dictionary information by using DBMS. Such CASE or DBMS interaction demonstrates the interdependence that exists between system development and database development, and it helps create a fully integrated development environment.
9	Quality of communication [60]	A CASE environment tends to improve the extent and quality of communication among the DBA, application designers, and end users.
10	Cost and budget management [33,60]	CASE tools aim to address the difficulties of developing high-quality, complex software on time and within budget.
11	Integration across the platform [60]	The CASE tool integrates all system development information in a common repository, which can be checked by the DBA for consistency and accuracy.

CASE, computer-aided software engineering; SDLC, system development life cycle; DBMS, database management system; DBA, database administrator.

3. Research method

The overall research is approached as follows. The first stage (called preliminary work) has concerned a literature review and data collection on the state of PCS software development as well as CASE tools in general; it is explained in Section 3.1, but has already been presented in Section 2. The second stage focuses on the presentation of our view-based framework built in the CASE tool; it is presented in Section 3.2. Within these views, the PVM and CRC cards, as well as other complementary sources of knowledge, are conceptualised. Consistency between different views is ensured by the CASE tool's internal logic, and code can be generated from the knowledge represented in different views. The third and final stage is devoted to validating the value of the use of the CASE tool on case studies from two different industrial engineering companies; it is presented in Section 3.3. Figure 3 highlights these different stages of the research.

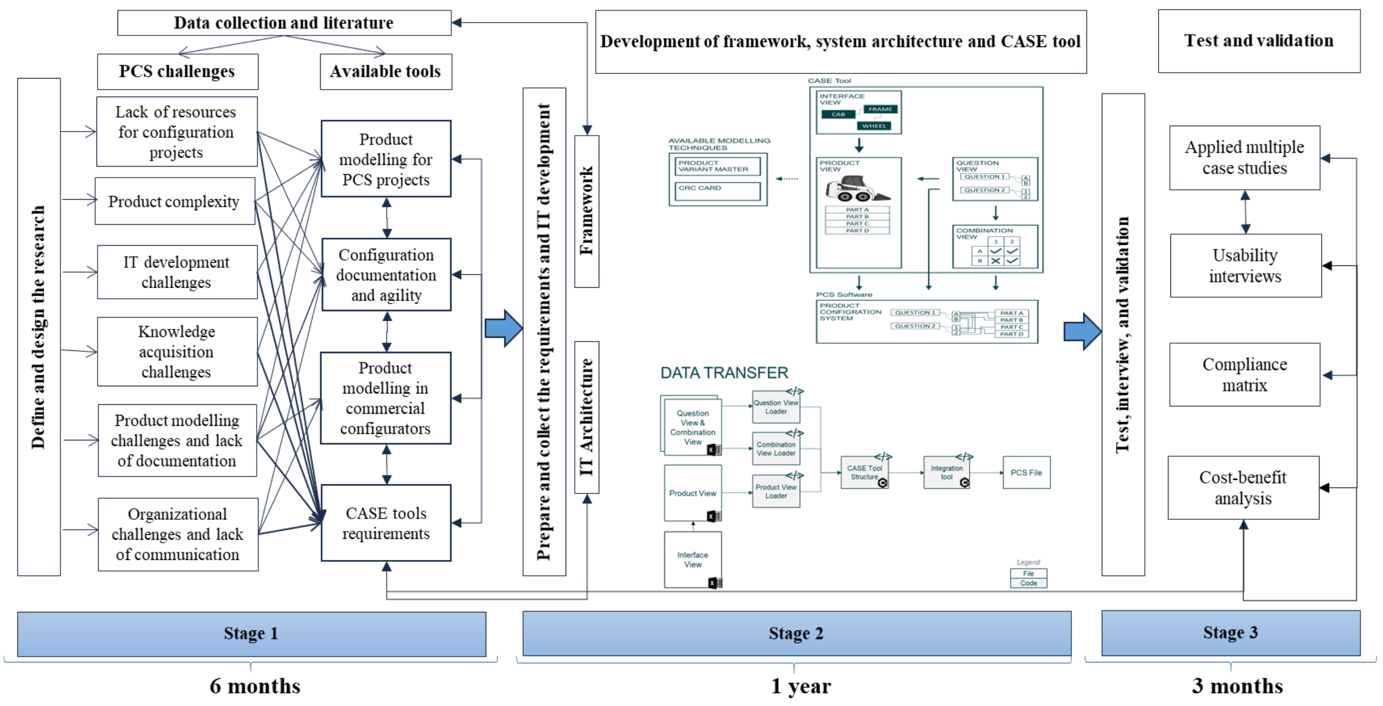

Data collection and literature
PCS challenges
Available tools
Define and design the research
Lack of resources for configuration projects
Product complexity
IT development challenges
Knowledge acquisition challenges
Product modelling challenges and lack of documentation
Organizational challenges and lack of communication
Product modelling for PCS projects
Configuration documentation and agility
Product modelling in commercial configurators
CASE tools requirements
Prepare and collect the requirements and IT development
Framework
IT Architecture
Development of framework, system architecture and CASE tool
CASE Tool
INTERFACE VIEW
CAB
FRAME
WHEEL
AVAILABLE MODELLING TECHNIQUES
PRODUCT VARIANT MASTER
CRC CARD
PRODUCT VIEW
QUESTION VIEW
COMBINATION VIEW
PCS Software
PRODUCT CONFIGURATION SYSTEM
DATA TRANSFER
Question View & Combination View
Question View Loader
Combination View Loader
Product View
Product View Loader
Interface View
CASE Tool Structure
Integration tool
PCS File
Legend
File
Code
Test, interview, and validation
Test and validation
Applied multiple case studies
Usability interviews
Compliance matrix
Cost-benefit analysis
Stage 1
Stage 2
Stage 3
6 months
1 year
3 months

Figure 3. The methodology for developing the framework and computer-aided software engineering (CASE) tool; application programming interface (API).

3.1. Preliminary work

Figure 3 shows the approach followed during the research. More specifically, it shows the stages followed for the development of the view-based framework, its supporting CASE tool, and then the results' analyses. The entire research took approximately two years. First, the literature study was conducted over one year with the aim of investigating the availability of CASE tools or any similar tool for automatic development of PCS projects. Accordingly, the PCS challenges, available tools, the relations between the challenges [8,21–24,27,29], and the potential solutions have been listed [1,19,28,49,51,57,62]. Moreover, we build the suggested CASE tool snippets based on the previously proposed solutions in the literature as well as the reported benefits from the CASE tools, as listed in Table 1 [33,59–61]. Hence, the goal of the proposed tool is to combine, integrate, and mainly automate model-driven development of PCS software based on the PVM and CRC cards while minimising the reported challenges [33,59–61].

3.2. Development of the view-based framework and the CASE tool

The view-based framework and the supporting CASE tool have been developed following an iterative process. We improved the framework and tool based on the feedback and comments from researchers, users, teams, and managers. Figure 3 shows that Stage 2 is essentially concerned with the CASE tool presentation; it is thoroughly presented and explained in Section 4; the views give the logical architecture of the tool (this is notably depicted in Figures 4-10).

3.3. CASE tool evaluation

Finally, in the last stage of the research, we tested our CASE tool in two different companies on two real case projects where we developed a real project for the case companies. Within that context, we conducted interviews and analysed our tool in a compliance matrix. We evaluated the proposed framework and CASE tool to determine: (1) if the structure, attributes, and constraints of commercial PCS software could be documented and maintained in the CASE tool to a full extent in a real-life project setting; (2) if the PCS software can be built completely using the CASE tool at the case companies; and (3) if development and maintenance of a PCS project is cost-efficient compared to the situation of absence of the CASE tool.

Usability is "the extent to which the IT system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use" [63]. Nielsen [64] lists five characteristics of usability. Using the definition of usability provided, we formulated the following interview questions:

1. How much time is required to learn the CASE tool and its user interface?
2. How much (resource) time is saved by using the CASE tool for development, compared with older methods, such as Excel spreadsheets and doing the modelling step by step in the PCS?
3. How much (resource) time is saved by using the CASE tool for maintenance, compared with older methods, such as Excel spreadsheets and doing the maintenance inside the PCS?
4. How much error reduction occurs in the PCS with the use of the CASE tool because of the saved hours in development?
5. What is the user's level of satisfaction and acceptance of the CASE tool?

In relation to questions 2 and 3, it should be noted that by time, we mean resource time that includes time taken by the configuration team and domain experts. Using these questions, we assessed the saved resource time from using the CASE tool. The saved resource time also affects the total project lead time; therefore, PCS development and maintenance is incredibly important. The data was collected by interviewing different stakeholders at the case company to reflect different expectations about the documentation that could be managed through the CASE tool. Employees at the two-case company were interviewed to assess the CASE tool, and we selected two employees from each company. They were selected from the configuration team and domain experts. Accordingly, two employees were selected from the configuration team from cases 1 and 2 with four and eight years of experience working with the PCS, respectively. Two employees were selected from domain experts from case companies 1 and 2 with ten and seven years of work experience, respectively.

As presented in Table 1, there are many ways in which CASE tools can contribute to software quality. A compliance matrix is provided in Table 6 to show the compliance of the developed CASE tool with the overview of applicable requirements in Table 1. This matrix formalises requirements verification with respect to the developed system, which directly uses the dimensions of Table 1.

Moreover, we calculate the costs and benefits to provide the cost savings results for researchers and practitioners from using the CASE tool. Cost–benefit analysis is used to compare the expected costs and benefits for different scenarios and the results from a variety of actions [65]. Return On Investment (ROI), which is commonly used as a cost–benefit ratio, is a performance measure used to evaluate the efficiency of a variety of investments [66], and it has been used to determine the profitability of PCS projects. In the ROI formula, the costs are subtracted from the total benefits to produce net benefits, which are then divided by the costs. Normally, ROI is estimated and calculated between a three-to-five-year period to provide more reliability for study [35,67].

4. CASE tool, supported representations, and forward engineering

The CASE tool is described in this section from a more technical and functional point of view so that the reader can exactly figure out how it works to support the PCS development process and automate some parts of it. This section first presents its general architecture and then describes its characteristics and use. It indeed proposes multiple complementary views required to consistently build a PCS following the tool's inner logic.

4.1. Main Architecture of the CASE tool

As specified earlier, the CASE tool has been designed to further support an existing, industry-adopted method for PCS development fully depicted in [1]. The former method's originality is to include the PVM and CRC cards as main artefacts for PCS analysis. These artefacts' knowledge is represented in the tool through complementary views. The main advantage of using the CASE tool is to keep the knowledge of the PVM and CRC cards consistent throughout the PCS development life cycle within the different views. Moreover, the CASE tool also guarantees the enrolment of various stakeholders—who also have various technical and non-technical expertise. These stakeholders are capable of changing PCS structure and, by sharing and editing the representations inside the CASE tool, consistency is ensured. The entire project team then keeps working on up-to-date representations of PCS knowledge. Also, in terms of maintenance, the use of the CASE tool ensures that relevant knowledge is archived and kept available for domain experts; changes in PCS structure after deployment can still be logically documented in the CASE tool for consistent documentation and version control. When enough knowledge is represented/included into the different views, the CASE tool can generate code (forward engineering).

Figure 4 summarises this structure/architecture. The PVM and CRC cards are seen as the existing conceptual basis for knowledge representation; the CASE tool's constituting views do allow us to logically represent product knowledge; then, a skeleton of the code of the PCS software can be generated. All the views and the code generation are depicted in more detail in the next section.

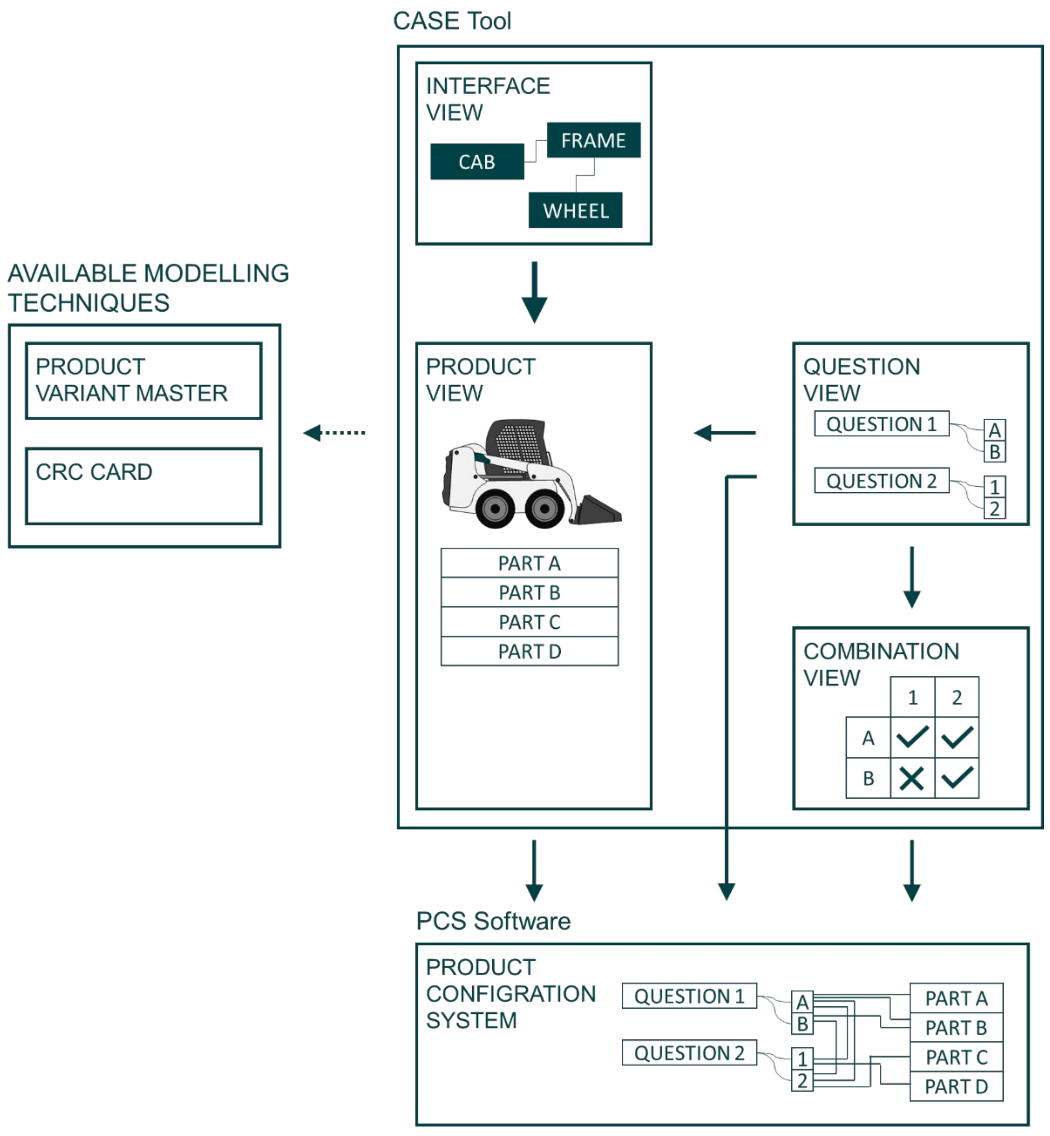


Figure 4. CASE-Tool Main Architecture: The plain arrows indicate the data flow within the IT Tool for modelling and to the commercial product configuration systems (PCSs); the example is a loader.

4.2. Characteristics and functions of the CASE tool

The complementary views allow management of the knowledge required to implement the PCS supporting software. More specifically, the tool distinguishes:

1. The *interface view* visualises and conceptualises the product structure and understands relations between modules. This view:
   - Aligns the product structure with the product view;
   - Describes the constraints of the product.
2. The *product view* to represent the product information. This view:
   - Provides the product structure retrieved from the interface view;
   - Provides product information (only control product information: component dimensions, engineering hours, colours, etc.);

- Explains and validates the product's detailed parts and components;
- Aligns different types of data with all other views and connects them together.

3. The *question view* to represent the PVM and CRC cards in an integrated manner. This view:
   - Creates the question structure and data (the questions provided for product configuration and user guidance through the configuration process).
4. The *combination view* to represent configuration data. This view:
   - Provides component combination (different ways that the components can be combined);
   - Provides constraint solution space (the feasibility and allowance for combining the components).

In this section, the CASE tool is further described through its different views, and each one is illustrated through an example. All the examples are aligned and from a specific product: a *loader* as a heavy equipment machine used to move aside or load various kinds of materials. The loaders are sold for different applications (e.g. forest with challenging terrain or construction sites where heavy lifts are important). The different applications create various requirements for the loader (e.g. requirements of engine output meaning different engine sizes, requirements of operating capacities meaning different buckets and hydraulic systems, etc.). Furthermore, it is only certain motors that can operate with certain operating capacities, e.g. running with high operating capacities with a small engine leads to engine failure. Hence, there are various and complex configurations of loaders. This loader product has been selected as an illustrative example because it is easy to understand but complex enough to challenge our CASE tool by offering many configuration possibilities. More complex case studies have also been implemented using it, but they are not depicted through technical details here because most of them are confidential, require the acquisition of a lot of domain knowledge, and would not necessarily allow understanding of the CASE tool's support and behaviour better than with a more trivial example as the loader. Specifically, the loader is a product like a real-life project; it aligns different types of data with all other views and connects them together. It has specific properties and constraints that must be made available or communicated across the entire team for the development and testing of configurations as well as the deployment in the real-life setting. These characteristics are illustrated in the context of the CASE tool in this section.

4.2.1. Interface view

The *interface view* sustains a conceptual model of the whole product structure (as can be seen in Figure 5); it aims to facilitate the understanding and validation of the product's overall structure. Typically, a PCS is constituted of different modules: in the case of the loader example, we can distinguish the *equipment*, *loader body*, and *base frame* as main modules. The modules are constituted of functional units that are combined to describe the configurable product model inside the PCS software. The interface view is a fundamental starting point for PCS development and management. All the modules can be edited (i.e. modified or updated) and changes will be automatically updated in other views.

As shown in Figure 5, the interface view can be used to manage the allocation of responsibility of different product modules to domain experts, furnishing information about the configurable product to be supported by the PCS software. This is normally done in the early phases of the project when we scope the PCS project. When changes are made and approved, the interface diagram is implemented with a version number.

The interface view is used especially in the initial phases of a PCS development project to define its conceptual structure. Then, through iterations in product development, different modules and views would gain maturity, which means their specifications would become clearer and more comprehensive. Knowledge is likely to evolve iteratively, and so details would be available later in the development life cycle.

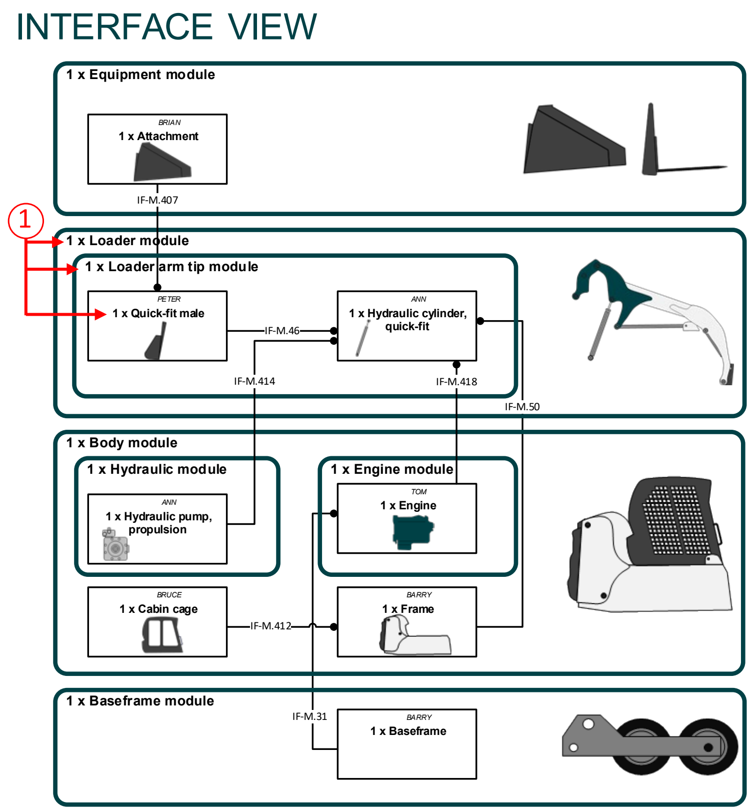


Figure 5. The interface view shows a *loader* that consists of an *Equipment module*, *Loader module*, *Body module,* and *Base frame module*. The number 1 illustrates the product structure (part tree): *1 x Loader module*, *1 x Loader arm tip module* and *1 x Quick-fit male*.

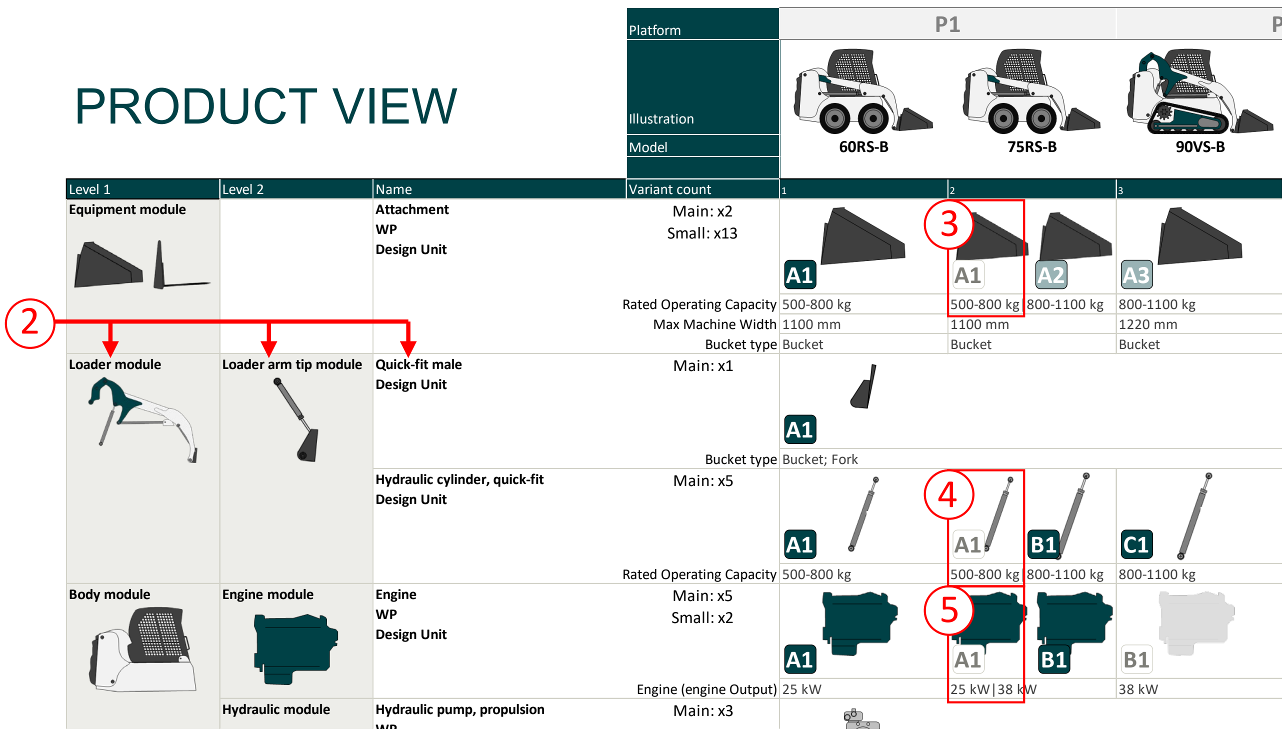
The different knowledge categories required for the interface view are provided in Table 2. These knowledge fragments can be designed by IT or non-IT experts working on the project.

Table 2. Required knowledge for the interface view

Different knowledge categories	Explanation
General knowledge included	• Manage the overview of the conceptual product structure
Included knowledge for development phase	• Create the product structure • Create the naming of modules and parts
Included knowledge for maintenance phase	• Update the product structure • Update the naming of modules and parts

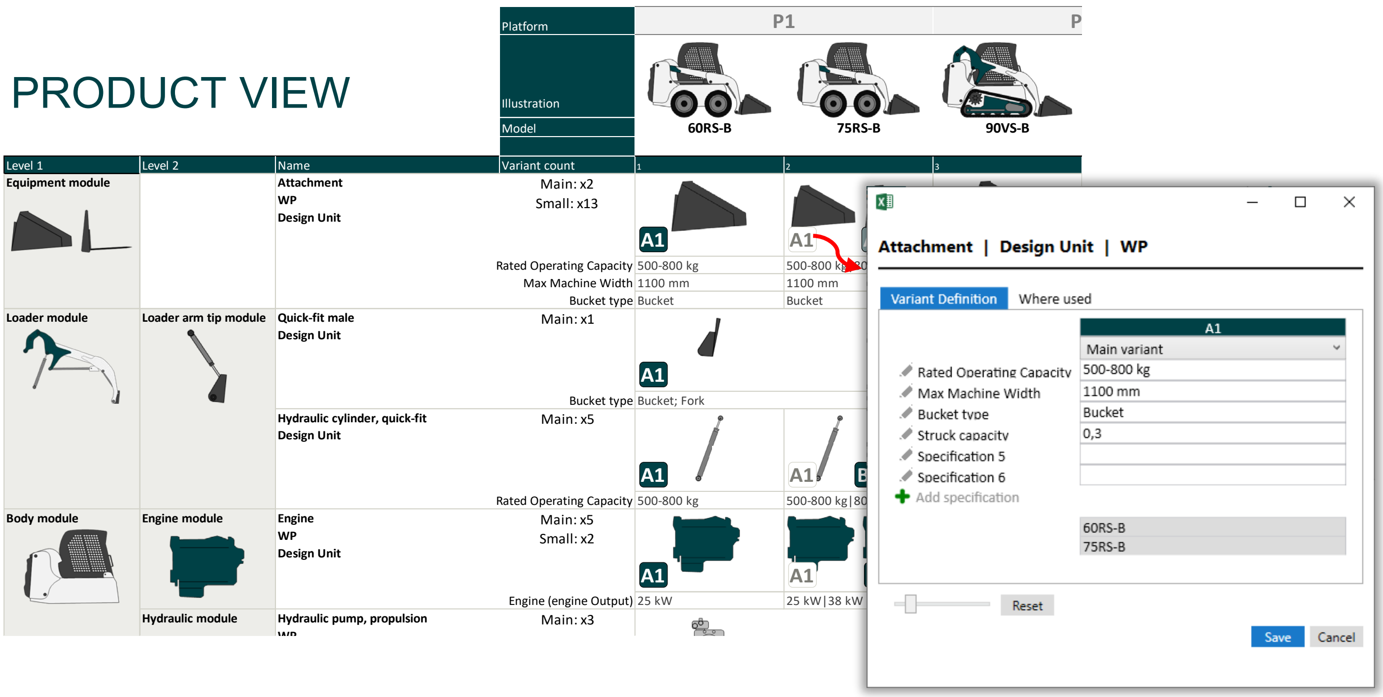
4.2.2. Product view

The product view can be considered the most important view of our CASE tool, as it infers knowledge from all other views provided in the CASE tool. The product view allows the product characteristics of a particular development project to be represented and further edited. The data included in this view helps manage the whole product portfolio by showing different properties of components, attributes, and features, including relevant representations. Figure 6 shows, on the left part, the product's hierarchical structure inherited from the interface view, similar to what was introduced by Hvam et al. [1]. The top part shows the different commercial models that can be made through multiple configurations. The product view consists of the *Design Unit Variants* (*DUV*), which consist of numerous specifications and are aligned to the CRC card structure, including attributes and figures. Figure 7 shows the DUV specification window, which facilitates the access and editing of the DUV specifications and assigned model. In this example, we see the DUV specifications of the attachment. Moreover, the domain knowledge related to the PCS required for developing the fully consistent product view is provided in Table 3. These different types of knowledge are provided and validated by domain experts in the field by filling out the product view.


PRODUCT VIEW
2
3
4
5

			Platform	P1		P
			Illustration			
			Model	60RS-B	75RS-B	90VS-B
Level 1	Level 2	Name	Variant count	1	2	3
Equipment module		Attachment WP Design Unit	Main: x2 Small: x13	A1	A1 A2	A3
			Rated Operating Capacity	500-800 kg	500-800 kg 800-1100 kg	800-1100 kg
			Max Machine Width	1100 mm	1100 mm	1220 mm
			Bucket type	Bucket	Bucket	Bucket
Loader module	Loader arm tip module	Quick-fit male Design Unit	Main: x1	A1		
			Bucket type	Bucket; Fork		
		Hydraulic cylinder, quick-fit Design Unit	Main: x5	A1	A1 B1	C1
			Rated Operating Capacity	500-800 kg	500-800 kg 800-1100 kg	800-1100 kg
Body module	Engine module	Engine WP Design Unit	Main: x5 Small: x2	A1	A1 B1	B1
			Engine (engine Output)	25 kW	25 kW|38 kW	38 kW
	Hydraulic module	Hydraulic pump, propulsion WP	Main: x3			

Figure 6. Product view—In the left part of the figure, the part structure (part tree) is inherited from the Interface View (Figure 5) explained by number 2. The top part shows the commercial models of the loader. The product view shows the Design Unit Variations (DUV) and their specifications for each model in numbers 3, 4, and 5.


PRODUCT VIEW

Platform	P1		P
Illustration			
Model	60RS-B	75RS-B	90VS-B

Level 1	Level 2	Name	Variant count	1	2	3
Equipment module		Attachment WP Design Unit	Main: x2 Small: x13	A1	A1	
			Rated Operating Capacity	500-800 kg	500-800 kg|80	
			Max Machine Width	1100 mm	1100 mm	
			Bucket type	Bucket	Bucket	
Loader module	Loader arm tip module	Quick-fit male Design Unit	Main: x1	A1		
			Bucket type	Bucket; Fork		
		Hydraulic cylinder, quick-fit Design Unit	Main: x5	A1	A1 B	
			Rated Operating Capacity	500-800 kg	500-800 kg|80	
Body module	Engine module	Engine WP Design Unit	Main: x5 Small: x2	A1	A1	
			Engine (engine Output)	25 kW	25 kW|38 kW	
	Hydraulic module	Hydraulic pump, propulsion WP	Main: x3			

Attachment | Design Unit | WP

Variant Definition | Where used

	A1
	Main variant
Rated Operating Capacity	500-800 kg
Max Machine Width	1100 mm
Bucket type	Bucket
Struck capacity	0,3
Specification 5	
Specification 6	
Add specification	
	60RS-B
	75RS-B

Figure 7. The product view of the CASE tool for the loader and product specification window for editing specifications and assigning DUV to the associated models. Specifications can be viewed and edited, and more can be added in the *Variant Definition* tab.

Table 3. Required knowledge for the product view

Different knowledge categories	**Explanation**
General knowledge included	• Product specification (attributes) • Prioritised product archetypes (hierarchical structure)
Included knowledge for development phase	• Build up prioritised solutions in the early phases of the project • Create the portfolio overview • Create the whole solution space just by using feasible combinations
Included knowledge for maintenance phase	• Maintain the product specifications (attributes) • Control different versions of the documented knowledge

4.2.3. Question view

The question view is concerned with the Graphical User Interface of the *to-be* PCS software (this is known as the execution part in the implemented PCS). The final user interface's inputs/outputs are designed in the question view, and this view is transferred to the execution part inside the PCS software.

All the knowledge presented in this view can be retrieved from other views. To configure a product, adequate questions must be provided and asked from the customers in the user interface. The decision of when and where each question about the product configuration should be asked of the user (so the process of configuring the product) is strongly dependent on product configurability options (Table 4). Figure 8 shows the customer-related requirements to offer feasible solutions using the available data and constraints. In this specific example, the question is about the loader's *Rated Operating Capacity*. In other words, we can have loaders with different Rated Operating Capacities depending on the loader range (product view) and which variants are decided to be offered. Customers are responsible for their own configuration among the provided options.

The CASE tool needs to specify which design units in the product view are impacted when you change an option in the question view. The tool makes the link between the question view and the product view when a specification and a question have the same name. For example, when a user makes a selection for the question ‘Rated Operating Capacity’ and selects the value 750 (Figure 8 No.6). The tool has a link to the product view where a specification has the same name as ‘Rated Operating Capacity’. The tool then finds the design units with this specification and locates the variant that can match the value 750. In this case, Attachment Variant A1 has a range of 500–800 (Figure 7 No.3). Moreover, the design unit ‘Hydraulic cylinder, quick-fit’ will be impacted by this (see question in Figure 7 No.4) because this also has the specification ‘Rated Operating Capacity’. Figure 6’s numbers 3 and 4 illustrate that these DUVs, which are used in the Rated Operating Capacity question (No. 6 in Figure 9) and is only compatible with a capacity between 500 and 800 kg.

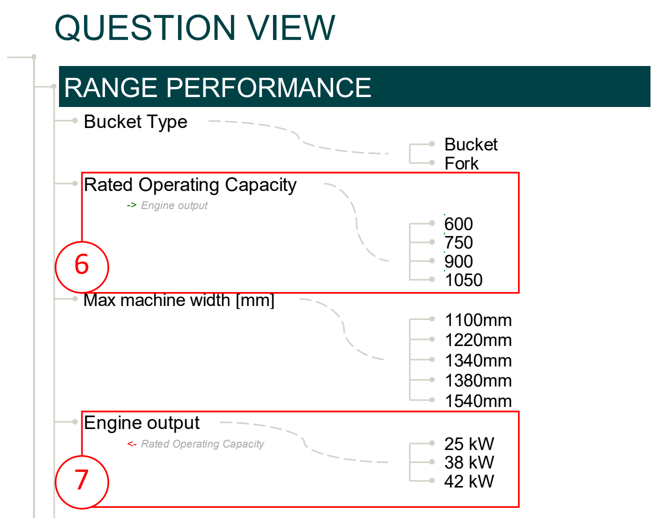


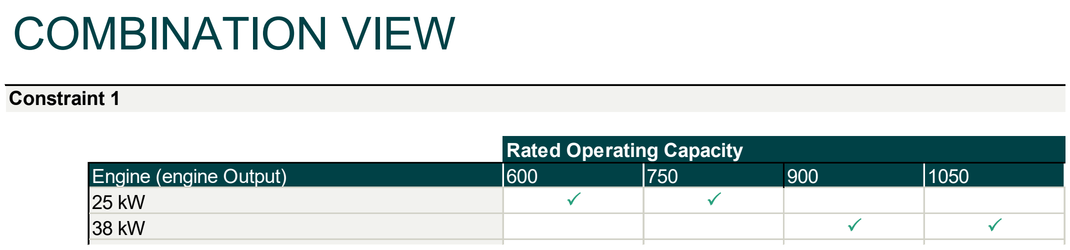
Figure 8. Question view—the example shows questions and options that the users are required to take to select the correct product and features; numbers 6 and 7 demonstrate the input options for *Rated Operating Capacity* and *Engine output*.

Table 4. Required knowledge for the question view

Different knowledge categories	Explanation
General knowledge included	• Manage the question structure and data • Manage the final user interface questions in the PCS
Included knowledge for development phase	• Create the questions • Specify the question structure • Specify the question data
Included knowledge for maintenance phase	• Create the questions • Update the question structure • Update the question data

4.2.2. Combination view

Typically, a PCS is made up of different components that need to be assembled together as different parts of a product or process and are interrelated, including both commercial or/and technical perspectives of a product [3]. The combination view contains constraints to combine different components together. An example of a constraint for the loader is between the engine output in Figure 8 number 7 and the Rated Operating Capacity in Figure 8 number 6. Since a small engine will collapse if it is exposed to high operating capacity, the models cannot be sold with a 25-kW engine output with a Rated Operating Capacity of more than 800. The combination view of Figure 9 ensures that only the correct combinations can happen. Figure 9 illustrates how the 25-kW engine can only be combined with the 600 and 750 Rated Operating Capacity, and the 38 kW can only run at a capacity of 900 and 1050. These constraints need to be documented explicitly for adequate management of the final PCS-related knowledge.


COMBINATION VIEW

Constraint 1

	Rated Operating Capacity			
Engine (engine Output)	600	750	900	1050
25 kW	✓	✓		
38 kW			✓	✓

Figure 9. Combination view—This is one example of many potential configuration matrixes used to make sure only the right options can be combined.

The different knowledge categories required for the combination view are provided in Table 5. These knowledge fragments are provided and validated by domain experts in the field by filling out the combination view.

Table 5. Required knowledge for the combination view

Different knowledge categories	Explanation
General knowledge included	• Manage the solution space using constraints
Included knowledge for development phase	• Create the configuration solution space
Included knowledge for maintenance phase	• Maintain the solution space (options and attributes) • Control different versions

4.2.5. Forward engineering possibilities provided by the CASE tool

Outside the edition of various diagrams allowing to define the PCS, the CASE tool provides the ability to generate the code that can immediately be used for PCS implementation. Figure 10 shows the architecture of the forward engineering possibilities provided by the CASE tool. The architecture consists of three data loaders that load data from the *question view*, *combination view,* and *product view* that all together contain data needed for the PCS software. Information retrieved in the three data views of loaders is collected in the CASE tool structure code as the main code of the CASE tool. This code combines the data from the three loaders and forwards the information to the Application Programming Interface (API). The API will now convert the data from the CASE tool to a format that can be written by the PCS software and saved as a PCS file. The objects of the CASE tool, the objects of the PCS, and the dataflow done by the API are shown in Figure 11.

The CASE tool implicitly incorporates two different structures. One that represents the structure for the Question and Combination Views (see Figure 11) and one that represents the structure for the product view. These two structures are managed by the API and converted into the structure of the PCS software, as shown in Figure 11.

After designing the product within the CASE tool, data integration follows between the CASE tool and the PCS, which has been implemented through an integration tool[2] and the API. The integration module converts

[2] A data integration tool is a software tool which is used to perform a data integration process on a data source and is designed based on data integration requirements. This tool performs transformation, mapping, and cleansing of data.

the data provided from the CASE tool into the PCS. Like the rest of the CASE tool, the integration tool is programmed in C#; this facilitates communication between the CASE tool and the PCS.

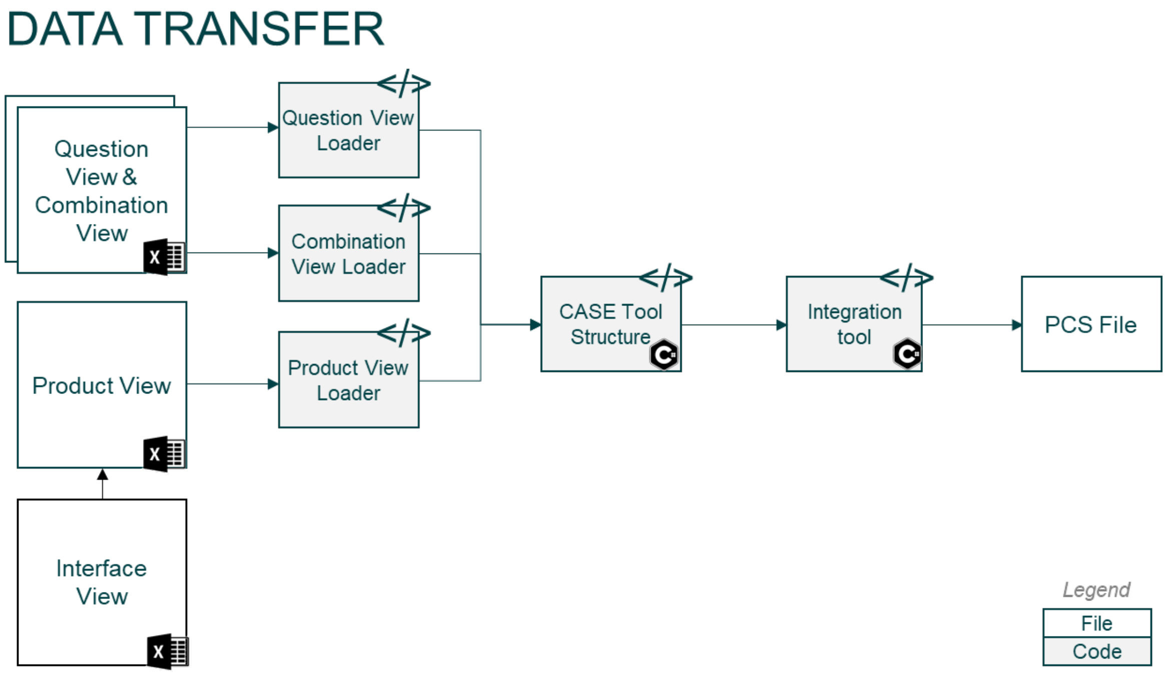


Figure 10. The configurator architecture showing different data flows from different CASE views into the CASE tool and integration tool, where the data is converted into a compatible configurator data format. API, application programming interface.

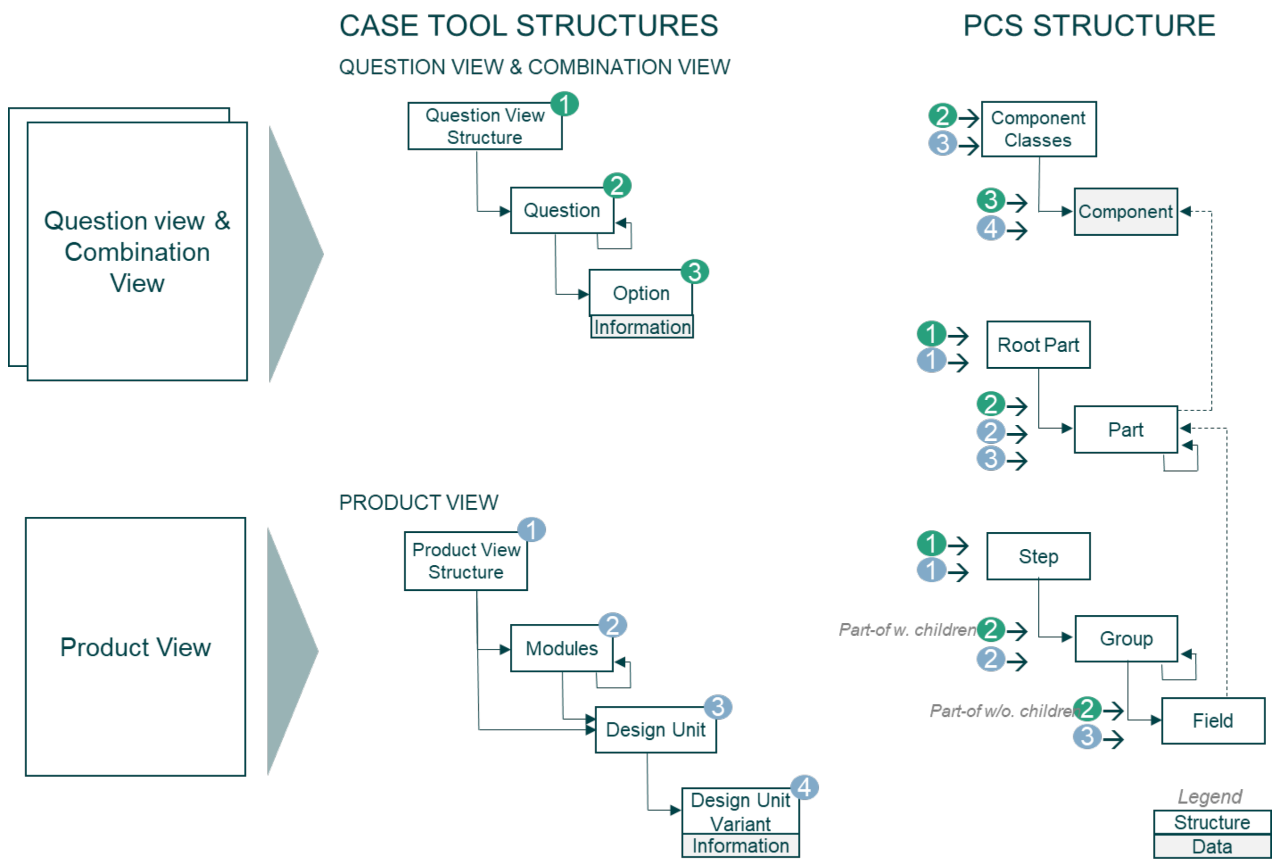
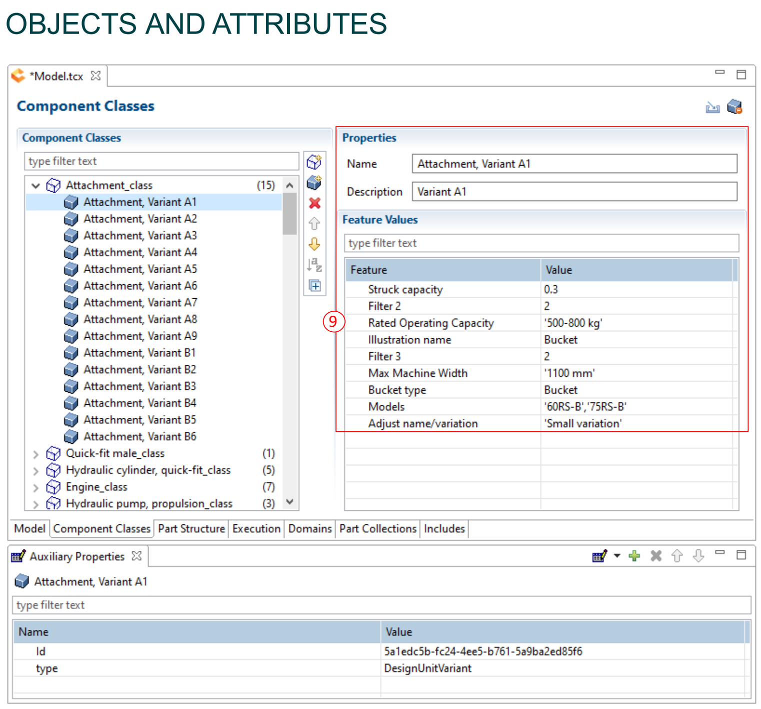


Figure 11. The architecture objects created in the different views and the correlation between the CASE tool structures and the PCS structure are shown.

Figures 12, 13, and 14 represent the PCS software's code and Graphical User Interface automatically generated from the CASE tool's knowledge on the basis of the different views. More specifically, Figure 12 demonstrates the product view. Figure 13 illustrates an example from the product view and its connection to the constraints in the combination view in the case tool that has been generated in PCS. It shows questions and options that the users are required to take to select the correct product and features. Finally, Figure 14 is the PCS generated based on the question view in the CASE tool to demonstrate the user interface. Moreover, Figure 14 presents the questions from the question view, which is connected to the data from the product view.


OBJECTS AND ATTRIBUTES

*Model.tcx

Component Classes

Component Classes

type filter text

- Attachment_class (15)
  - Attachment, Variant A1
  - Attachment, Variant A2
  - Attachment, Variant A3
  - Attachment, Variant A4
  - Attachment, Variant A5
  - Attachment, Variant A6
  - Attachment, Variant A7
  - Attachment, Variant A8
  - Attachment, Variant A9
  - Attachment, Variant B1
  - Attachment, Variant B2
  - Attachment, Variant B3
  - Attachment, Variant B4
  - Attachment, Variant B5
  - Attachment, Variant B6
- Quick-fit male_class (1)
- Hydraulic cylinder, quick-fit_class (5)
- Engine_class (7)
- Hydraulic pump, propulsion_class (3)

Properties

Name	Attachment, Variant A1
Description	Variant A1

Feature Values

type filter text

9

Feature	Value
Struck capacity	0.3
Filter 2	2
Rated Operating Capacity	'500-800 kg'
Illustration name	Bucket
Filter 3	2
Max Machine Width	'1100 mm'
Bucket type	Bucket
Models	'60RS-B','75RS-B'
Adjust name/variation	'Small variation'

Model | Component Classes | Part Structure | Execution | Domains | Part Collections | Includes

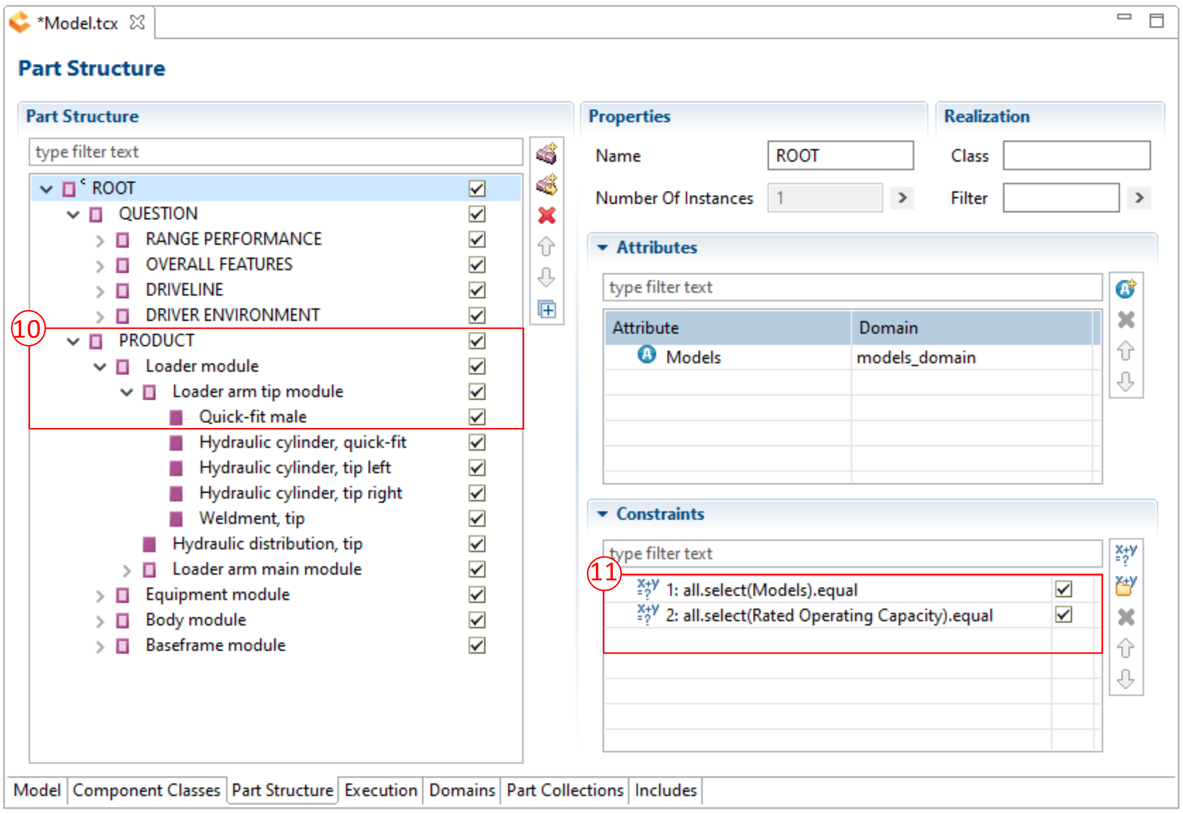
Auxiliary Properties

Attachment, Variant A1

type filter text

Name	Value
Id	5a1edc5b-fc24-4ee5-b761-5a9ba2ed85f6
type	DesignUnitVariant

Figure 12. The objects from the CASE tool created in the PCS—The example illustrates the data from Figure 6 No. 3 being the same data indicated by No. 9.


*Model.tcx

Part Structure

Part Structure

type filter text

- ROOT ✓
  - QUESTION ✓
    - RANGE PERFORMANCE ✓
    - OVERALL FEATURES ✓
    - DRIVELINE ✓
    - DRIVER ENVIRONMENT ✓
  - PRODUCT ✓
    - Loader module ✓
      - Loader arm tip module ✓
        - Quick-fit male ✓
        - Hydraulic cylinder, quick-fit ✓
        - Hydraulic cylinder, tip left ✓
        - Hydraulic cylinder, tip right ✓
        - Weldment, tip ✓
      - Hydraulic distribution, tip ✓
      - Loader arm main module ✓
    - Equipment module ✓
    - Body module ✓
    - Baseframe module ✓

10

Properties

Name	ROOT
Number Of Instances	1

Realization

Class	
Filter	

Attributes

type filter text

Attribute	Domain
Models	models_domain

Constraints

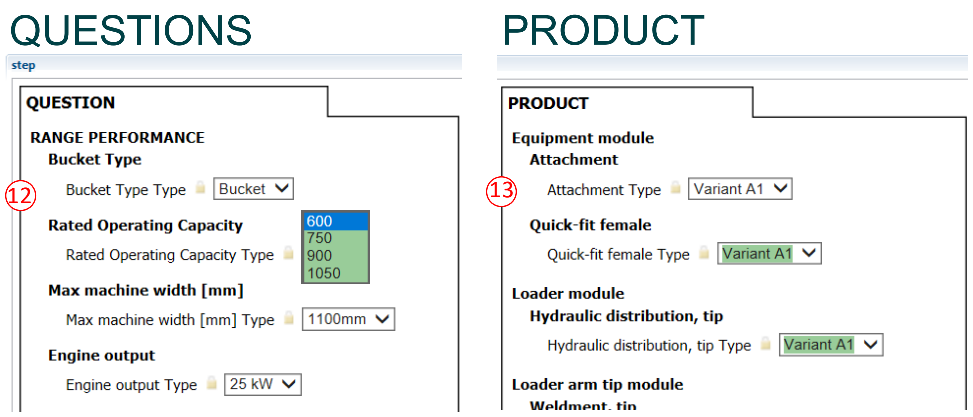
type filter text

11

1: all.select(Models).equal	✓
2: all.select(Rated Operating Capacity).equal	✓

Model | Component Classes | Part Structure | Execution | Domains | Part Collections | Includes

Figure 13. The example shows the inherited product structure No. 10 from the product view Figure 6 No. 2, and constraints No. 11 inherited from the combination view Figure 8 together with the obligatory constraint all.select(Models).equal.


QUESTIONS

step

QUESTION

RANGE PERFORMANCE

Bucket Type

(12) Bucket Type Type: Bucket

Rated Operating Capacity

Rated Operating Capacity Type: 600 / 750 / 900 / 1050

Max machine width [mm]

Max machine width [mm] Type: 1100mm

Engine output

Engine output Type: 25 kW

PRODUCT

PRODUCT

Equipment module

Attachment

(13) Attachment Type: Variant A1

Quick-fit female

Quick-fit female Type: Variant A1

Loader module

Hydraulic distribution, tip

Hydraulic distribution, tip Type: Variant A1

Loader arm tip module

Weldment, tip

Figure 14. The human machine interface in test mode showing the *questions* and the *options* No. 13. from the question view from the CASE tool Figure 9. The product view also in test mode showing the inherited data from the product view Figure 7, including the product structure with parts No. 14 and the part variant: "Variant A1".

4.3. Compliance matrix: Support of typical CASE-Tool features

We use, in this section, a compliance matrix to evaluate to what extent the CASE tool supports the features identified and discussed in Section 2.4. Table 6 thus shows the compliance of the CASE tool developed with the typical features found in CASE-tools as identified in Table 1. We see that most of the features are fully or partially supported by our CASE-tool so all in all it supports a model-driven engineering approach and can be considered as a consistent and self-contained tool for the support of PCS software development.

Table 6. Compliance matrix for the verification of requirements with respect to Table 1 and the proposed CASE tool

	PCS CASE tool compliance matrix				
	Features	**Compliance**			**Comments**
		Full	**Partial**	**None**	
1	Document handling	✓			Replacement of PVMs, CRC cards, Excel spreadsheets, and many other ways of documentation and obviously no duplication of knowledge.
2	Individual preparation	✓			There are various levels of integration, from simply reporting defects to producing an annotation related to the defect for the reviewer to examine.
3	Data collection	✓			Efficient knowledge management, knowledge validation, and knowledge maintenance.
4	Automation of the SDLC		✓		Support for the planning, analysis, and design phases; support for the coding and implementation phases.
5	Easier maintenance of application systems	✓			Efficient documentation and ease of access to knowledge make the maintenance more efficient; better

					communication and overview for experts and project members.
6	Standardisation of system development methodologies	✓			Keep track of the needed data and automatic modelling.
7	Data dictionary	✓			The CASE data dictionary stores all product knowledge in the PVM and inside the PCS, including components, attributes, constraints, and rules.
8	Interaction with DBMS			✓	The CASE tool is in interaction with the PCS, but there is no interaction with other DBMS tools.
9	Quality of communication	✓			The communication improvement and efficiency among the configuration team, developers, testers, domain experts, managers, etc.
10	Cost and budget management	✓			The CASE tool removes difficulties of developing high-quality, complex PCSs on time and within budget.
11	Integration across the platform		✓		The CASE tool integrates only with the PCS for now.

PVM, product variant master; CRC, class responsibility collaboration; CASE, computer-aided software engineering; PCS, product configuration system; DBMS, database management system.

5. Evaluating the CASE tool: case studies

5.1. Presentation of case studies

The proposed approach (the framework and CASE tool) was evaluated in two PCS projects from two selected engineering manufacturing companies. The first case company is focused on the wind industry as a leader within the field of nacelle and spinner covers. They are producing highly engineered complex products and services, and the core business is to design, and supply nacelle and spinner covers, with maximum customer benefit at minimum cost. This PCS is designed for customers to enable them to configure their own product, focusing on one of the main components with high frequency in maintenance. The second case company, which operates globally, specialises in catalyst production and process plant technology. Hence, this case company also offers engineered complex products in the field of chemical engineering. The selected project focuses on the spare parts of one of the most-sold plants. The profit out of spare parts is significant, and the company aims to use a PCS internally to do fast and efficient engineering and provide customers with fast and high-quality service.

Both companies reported difficulties with the development and maintenance of their PCSs, as well as a lack of documentation to support communication with domain experts. We assume that if the proposed framework can solve such challenges at one company, other companies with the same level of complexity, or with less complex products, might also benefit from it. A project team was formed at the case company, including two full-time researchers from the university, together with a software developer from the company, who spent approximately half of their time working on the project over the last year. The major role of the researchers was to develop the proposed approach, extract and analyse knowledge, prioritise the requirements of the CASE tool, and test and facilitate interviews to evaluate the models generated by the system. A software developer aligned with the configuration engineer programmed the CASE tool. The total effort required to build the CASE tool was calculated to be 1,000 man-hours, which included performing requirement analysis, designing the architecture of the system, developing the CASE tool (excluding building up the PCS), and testing, and training. It is estimated that the CASE tool had a positive ROI in the first year because of saved time and resources in different stages of the configuration projects. The CASE tool was used during development and operation. The characteristics of the projects are provided in Table 7.

For each of the projects, workshops were held to demonstrate the CASE tool value and obtain feedback. The content of the workshops included the introduction of the CASE tool, training on the use of the tool, and an open-forum segment that encouraged feedback and suggestions for improvement, more information can be found in Appendix A.

Table 7. Background information for the project implemented in the case studies using the CASE tool

Projects	Project 1	Project 2
Time frame for development (man-days)	20	145
Time frame for implementation (man-days)	5	20
Time frame for maintenance (man-days)	5	15
Complexity of the project	Small	Medium–large
Time for full-time resources assigned in each project	3 resources (core team)	10 resources (core team)
Number of workshops	2	1 every second week
Number of feedback meetings	2	1

In this study, parameter complexity is important, as it measures the complexity involved in providing configuration data to the computer system during a configuration procedure [68]. Therefore, we assessed the parameter complexity in terms of two major parameters inside the PCS: attributes and constraints (Table 8).

Table 8. Complexity assessment in terms of parameters in the product configuration system

Complexity	No. attributes	No. constraints
Small	500–1300	200–800
Medium	1300–2000	800–1200
Large	>2000	>1200

5.2. Evaluation of the CASE-Tool on case studies

In this section, we will evaluate the CASE tool. First, we will present and discuss the results from the usability test. Second, we demonstrate the compliance of the developed CASE tool and an overview of the applicable requirements for the CASE tools. Third, a cost–benefit analysis was performed in five years for configuration projects' cost and benefits using the CASE tool compared to the traditional development procedure. Finally, a few of the CASE tool benefits are mentioned according to the overall results from case studies.

5.2.1. Usability Evaluation

Employees at the case company were interviewed to assess the CASE tool. They were selected from the configuration team and domain experts. The main results from the interviews are summarised in Table 9.

Table 9. Usability interviews

Questions	Configuration team	Domain experts
How much time is required to learn the CASE tool and its user interface?	0.5–1 day	1–2 days
How much (resource) time is saved by using the CASE tool for development, compared with older methods, such as the PVM and doing the modelling step by step in the PCS?	30%–80% of the total time	30%, compared with the normal PVM and meetings
How much (resource) time is saved by using the CASE tool for maintenance, compared with older methods, such as the PVM and doing the maintenance inside the PCS?	30%–75% of the total time	75%, compared with the normal PVM and meetings
How much error reduction occurs in the PCS with the use of the CASE tool because of the saved hours in development?	80%	N/A
What is the user's level of satisfaction and acceptance of the CASE tool?	High	High

CASE, computer-aided software engineering; PVM, product variant master; PCS, product configuration system.

The usability test consists of five interview questions to assess the CASE tool in the case companies, as explained in Section 3.3. The differences in the answers depend on the interviewees' familiarity with PCSs and PVMs or CRC cards. The results from interviews in Table 9 indicate that the interviewees found the learning processes and use of the CASE tool easier than the previous communication or updating systems, such as Excel spreadsheets. The findings regarding time saving vary depending on product complexity. However, interviewees highlighted time savings by using the CASE tool. The configuration of the CASE tool prevents the overwriting of knowledge and reduced errors in the PCS because domain experts had control of the knowledge compared with the previously used systems. The level of detail considered in this CASE tool is not comparable with the PVM, as it includes all the details from the PVM plus all the requirements for PCS development. Therefore, once the CASE tool is filled out by domain experts, the configuration system is also developed. Furthermore, the results from the interviews indicate that (1) (human) resource effort is saved because of the efficiency and accessibility of the new tool to both IT and domain experts; (2) the quality of the PCS is improved because of good communication and verification of knowledge in a user-friendly CASE tool; and (3) the level of acceptance or satisfaction is high because of a friendly user interface and access to the knowledge for testing and debugging, as well as the easy automatic modelling of the PCS. Moreover, the IT department faces fewer organisational challenges and resistance to configuration systems because of the direct domain expert's contribution and ownership. This system provides users with knowledge documentation, validation, modelling, and communication with domain experts and the configuration team.

5.2.2. Cost–benefit analysis

To evaluate the benefits of the CASE tool, a cost–benefit analysis was performed. Table 10 presents the main results for the cases in this phase.

Table 10. Results of the cost–benefit analysis in 5 years using the CASE tool for configuration projects compared to the traditional development procedure

		Traditional development and maintenance	CASE tool development and maintenance
Case 1	Investment cost: software and development hours (including training and implementation) (hours)	100	20
	Annual costs: software and maintenance hours (hours)	3000	300
	Total project cost in 5 years (hours)	3100	320
	Total project benefits (comparing traditional PCS management and CASE tool management): 2780 hours CASE tool investment: 1000 hours		
	ROI in 5 years: 178%		
Case 2		**Traditional development and maintenance**	**CASE tool development and maintenance**
	Investment cost: software and development hours (including training and implementation) (hours)	1650	950
	Annual costs: software and maintenance hours (hours)	1700	900
	Total project cost in first year (hours)	3350	1850
	Total project benefits (comparing traditional PCS management and CASE tool management): 1500 hours CASE tool investment: 1000 hours		
	ROI in 5 years: 50%		

PCS, product configuration system; CASE, computer-aided software engineering; ROI, return on investment.

We quantified the cost savings for a five-year period and calculated the ROI. Both projects have been developed and implemented in the last year. Hence, we estimate all the costs and benefits for a five-year period to provide ROI with higher validation. However, the only cost that needs to be estimated is the maintenance cost, which is a small part of yearly costs. The only benefit considered from the CASE tool was the saved hours, and not the quality or other qualified benefits. Moreover, the estimation for benefits is again regarding maintenance efforts, which have a small portion in our calculations. This benefit was calculated as the difference between man-hours invested for a configuration project, with and without the application of the CASE tool proposed in this study. The total cost was calculated as the project cost, which included the development, implementation, and yearly running costs, such as licences and maintenance activities. We settled on five years as the most commonly used period of time [67]. The ROI for the first project was considerable—the reason being the project was not very complicated and PCS development was just about modelling and coding all the data from the product and constraints, communication between stakeholders, validation, delegation of responsibility to the right people, etc. Hence, in case 1, data could be gathered and developed very efficiently using the CASE tool, whereas in case 2, the project was more complicated and more time was spent on discussions, modularisations, and integrations, which are not covered by CASE tools and have to be done manually.

5.2.3. Workshops: lessons learned

As described in Section 5.1, workshops have been organized within each of the companies hosting the case studies. Feedback was indeed given by the stakeholders having an interest in the use and support of the CASE-tool for the PCS software development. From these discussions, various benefits were identified. The results are thus based on both of the PCS development projects supported by the CASE tool. The case studies' details as well as the number and duration of feedback meetings/workshops for each case project have been elaborated in Section 5.1,Table 7 and Appendix A. We here highlight the following CASE-too benefits (so as identified by the case studies' stakeholders) as being very relevant:

1. *A single version of the truth*. Using the CASE tool ensures that for similar structural parts and modules of the PCS software the aligned terminology is used. This constitutes a real advancement in terms of communication and potential use of the produced knowledge;
2. *Transparent and consistent data*. If any member of the PCS development team makes any change to the data of the PCS software directly in the CASE-tool, the changes are implicitly shared with the entire team. This way everyone is always working on an up-to-date version of the data
   Similarly, since the CASE tool is available to all stakeholders, everyone can provide, update, and maintain knowledge which makes collaboration more efficient. If any role changes any data in product view, all the changes will be implemented in PCS software automatically.
3. *Relevant views for every stakeholder*. The responsible experts (i.e. stakeholders classified in roles) for any part and stage of engineering of the product are capable of inserting and managing the knowledge relevant for their activities without having to deal with elements for which they do not have the technical expertise.
   This also allows each stakeholder to focus on its tasks and functions while enhancing and improving PCS software knowledge in the CASE-tool. For example, the tasks of inserting and validating product marketing knowledge can be assigned to the relevant experts.
4. *Improved communication with domain experts*. PCS software has specific properties and constraints that must be made available or communicated across the entire team for the development and testing of configurations as well as the deployment in the real-life setting; the CASE-tool definitely helps with this;
5. *Improved ability to produce documentation*: the CASE tool helps documenting, updating, integrating, and implementing the knowledge constantly and automatically.
6. *Full PCS software development life cycle support.*: The CASE tool supports each engineering stage from the knowledge representation in the four different views to code generation.

6. Threats to validity

This section studies the threats to validity with respect to our results; it is organised via three aspects: construct validity, internal validity, and external validity.

Construct validity reflects to what extent the operational measurements that are studied really represent what the researchers have in mind and what is investigated according to the research questions [69,70]. The main threat to construct validity is that the usability test interview questions are not interpreted by the interviewees in the way the interviewers intend. To deal with this issue, observer triangulation has been implemented [69].

Concretely, the research team that performed the interviews (but also some document analysis) consisted of two researchers.

Internal validity 'is the extent to which we can establish a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships' [69,70]. The major threat to internal validity is that the successful representation of the CASE tools in the literature and the automation of the whole process might compel the researchers to over-emphasise elements of a result set that supports the research question. To deal with this threat, we have linked the CASE tool functions and the study results to standard elements of the CASE tools emphasised as important in the literature.

The threat to *external validity* concerns the fact that the results may not be generalizable beyond the two investigated case studies [69,70]. Even though the findings of this study are only based on two different case companies, they have been chosen to be representative of typical PCS development projects in terms of PCS complexity (e.g. product options) and effort spent to increase the potential generalizability of the findings.

7. Discussion

The CASE tool enables the software development team, including IT professionals and domain (product) experts, to document, edit, and maintain product data in four different views. First, the interface view connects three different views of product information as one combined view. Second, the product view contains all the knowledge from all modules' details and predefined variants. Third, the combination view limits the solution space by using constraints and rules and modules' combinations. Fourth, the question view helps the user manage the user interface regarding the recommendation and guidance inside the configuration system.

The CASE tool provides a user-friendly IT environment with different categories to let users insert the knowledge in relevant pages. Moreover, we used the integration tool to link the CASE tool to PCS software (back and forth) through code generation to send and receive the knowledge. Finally, we discussed the whole approach in workshops and implemented the final suggestions and feedback. The case companies used and tested the CASE tool while developing and maintaining PCS software development projects for more than one year. The only challenge observed was regarding the training of the CASE tool and the needed workshops and time investment.

We now aim to summarise the elements overviewed in the paper to answer the RQ set in the beginning of the article, i.e.: *What are the concrete benefits of using a CASE tool to support (and partially automate) the model-driven development of PCS software to configure physical products?*

Four steps were referenced in the introduction in order to answer the research question; we take back these steps here and point out the findings.

First of all, thanks to a literature review we have identified the typical features found in CASE-tools, i.e. document handling, individual preparation, data collection, automation of the SDLC, easier maintenance of application systems, standardisation of system development methodologies (automatic performance), data dictionary, interaction with DBMS, quality of communication, cost and budget management and integration across the platform.

Second, we have developed a CASE-tool supporting the model-driven development of PCS software. The tool has been developed to support a custom framework divided in multiple views; it is based on existing techniques like the PVM, CRC cards and refined UML models and enhances traceability to ensure consistency; these views and their interactions have been fully described.

Thirdly, the developed CASE-tool has been crossed with the typical features that were found in CASE-tools through the literature review. To this end we have used a compliance matrix analysis. This matrix formalises the support of typical features by our CASE-tool. It allowed to identify that all features but the interaction with a DBMS are fully or moderately supported by our CASE-Tool. This has been evaluated and briefly justified in the paper.

Fourthly, the CASE-tool has been used for the full support of PCS software development projects in two selected engineering manufacturing companies; the selection criteria for the case studies have been explained. These two case studies were studied through several aspects. We started with a so-called usability study conducted on the basis of interviews. The results from the interviews indicate that (1) development effort is reduced because of the efficiency and accessibility of the new tool to both IT and domain experts; (2) the quality of the PCS is improved because of good communication and verification of knowledge in a user-friendly CASE tool; and (3) the level of acceptance or satisfaction is high because of a user friendly interface and access to the knowledge for testing and debugging, as well as the easy automatic modelling of the PCS. Then, we have estimated all the costs, benefits, and ROI for a five-year period to evaluate the value brought by the CASE tool. The ROI for both project are significantly positive. The ROI for the first project was considerably high; the project was focused very much on activities immediately supported by the CASE-tool. These were engineering activities (modelling and coding all the data from the product and its constraints) but also communication between stakeholders, validation, delegation of responsibility to the right people, etc. In the second case study, the project was involved more activities that are not immediately supported by the CASE tool like informal discussions, modularisations and integrations, the ROI was thus lower. Finally, workshops have been organized at the two case companies during the projects using the tool, results showed that it brought several benefits, namely a single version of the truth (aligned terminology is used for similar structural parts and modules), transparent and consistent data (shared data allows to keep it up to date at any moment), relevant views for every stakeholder (everyone can deal with its specific part of the software problem), improved communication with domain experts (technical data is easily spread), improved ability to produce documentation (through some of its features) and the full PCS software development life cycle support (from analysis to code).

8. Related Work

There are various challenges in developing and maintaining PCS, including the lack of resources, product-related challenges, IT difficulties, knowledge acquisition barriers, product modelling, and organisational challenges [8,21–24,27,29]. The importance of designing and developing a CASE tool for PCS software development projects lies in several factors [33,59–61]: (1) minimising the number of IT experts; (2) efficient, in control, automatic, and easy knowledge management (including knowledge acquisition, knowledge validation and testing, and knowledge maintenance); (3) transparent, aligned product data; (4) simultaneous documentation, communication, development, and validation of the product data; and (5) improved quality in both product knowledge documentation and configuration system. Indeed, even if past research calls for specific solutions and tools for PCS development to solve challenges, there is still a critical need for a supporting solution [71]. Industry and academics are constantly looking for new tools and techniques to deal with these challenges.

This paper has presented a view-based CASE tool for the model-driven development of PCS software which has been developed for the willingness of easing the development of such software so implicitly to deal with some of the aforementioned challenges. We used available modelling techniques for PCS communication and documentation [1,2,28,48,51,54,57]. To the best of the authors' knowledge no other CASE-tool for the development of software supporting the configuration of (physical) products do exist.

Bashroush et al. [40] have studied all the CASE-tool for the configuration of software product lines; our CASE-tool does deal with the same variability characteristics as these product lines essentially because in the manufacturing of physical products the dependencies do bring heavier constraints than for software; CASE-tools for the configuration of product lines thus tend to be less constraining in the proposed configurations. Applying our framework and CASE-tool in such a context is nevertheless planned for future work.

We can also compare our work with general CASE-tools. Lots of tools exist for the edition of UML models or workflow notations. We can for example cite Visual Paradigm [72] or Rational Software Architect [73]. These CASE-tools do perform very well for general software developments but physical PCS software development demands not only refinements in the analysis models but also in the UML design models. They are consequently not usable for a full life cycle support but can only help for some specific analysis or design tasks.

Other approaches have been used to evaluate CASE tools in literature. Teruel et al. [74] for example evaluate a case tool for requirements engineering on the basis of the user's ability to understand the tool and build a relevant representation with it. We were driven by other considerations in this paper. Since the tool has been used immediately by professionals with the support of a consulting team, we aimed to evaluate its ability to bring a ROI and support the entire life cycle of a PCS software development. We could nevertheless also study its capacity in the future to be understood by novice modellers with the use of students.

9. Conclusion

Considering Zeng et al. [32], this research contributes to all four aspects of CASE-based literature, i.e.: (1) modelling and representation of engineering information by effectively supporting the extraction of semantics from commercial PCSs; (2) conceptual structure of engineering information in forms of product/process structure that embodies the semantics; (3) algorithms for transforming different representations of engineering information into semantic conceptual structures by using code generation to be able to send and receive the knowledge; and (4) innovative application of CASE tools for PCSs as the first study to introduce the tool to support the development and implementation of PCS projects by researchers and practitioners.

This paper has presented a CASE-tool for PCS software development and validated its benefits on two case studies. The results of this study can be used as guidelines for companies and managers interested in further supporting PCS development when the knowledge that has to be encompassed in the software gets very complex. When implementing PCS software, companies should focus on the business cases to achieve the full potential benefits, while facing the challenges and probabilities of project failures. Based on the results from this research, using a relevant CASE tool not only automates the development and implementation of PCS projects to reduce the investment costs, but also will help dealing with the challenges related to PCS that can guarantee PCS success.

We point to a few limitations of the study that we have made. First of all, a lot of support has been provided within the case studies for the adoption of the CASE tool. We would also be willing to study how it can be adopted in PCS projects for teams who adopt it without the help of a consultancy team. To this end, we will perform a few experiments on novice modellers to see how easily they can analyse and design PCS software with the help of the CASE tool with only a paper-based documentation guidance. Also, the ergonomic aspects of the CASE tool can be further studied. Even if users have acknowledged on the ease of use, we could further optimize the setting of user interfaces to provide an even more efficient solution. This will be studied though ergonomic tests with a team specialized in the field. Finally, we aim to also further study the performance of the CASE tool when various software development life cycles are used like plan-driven and agile. The aim is to evaluate if more formal project management features could and should be included in the tool.

Future studies can focus on case companies with different customisation and personalisation development stages. Suzić et al. [75] introduces examples of configurators to challenge the linear implementation process of customisation enablers. To elaborate, because PCS was normally applied only once, modularity and product standardisation has been implemented, future studies can challenge the benefits of CASE tools to implement PCS projects even before the implementation of customisation enablers. We, therefore, suggest that future research should test the system using different projects, companies, types of software, and platforms such as small and medium enterprises. As the evaluation of the CASE tool requires not only the availability of the company, but also considerable time and resources, we only tested the approach at two companies. Focusing on two case companies, however, allowed us to gain an in-depth understanding of how the framework operates, as well as provided a detailed loop of improvements and simplifications required in the development of the system.

References

[1] L. Hvam, N.H. Mortensen, J. Riis, Product customization, Springer-Verlag, Berlin Heidelberg, Germany, 2008. doi:10.1007/978-3-540-71449-1.

[2] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based configuration from research to business cases, Morgan Kaufmann, Newnes, NWS, Australia, 2014. doi:10.1016/B978-0-12-415817-7.00029-3.

[3] C. Forza, F. Salvador, Product information management for mass customization: Connecting customer, front-office and back-office for fast and efficient customization, Palgrave Macmillan, New York, NY, 2006.

[4] A. Haug, S. Shafiee, L. Hvam, The causes of product configuration project failure, Computers in Industry. 108 (2019) 121–131. doi:10.1016/j.compind.2019.03.002.

[5] P. Zheng, X. Xu, S. Yu, C. Liu, Personalized product configuration framework in an adaptable open architecture product platform, Journal of Manufacturing Systems. 43 (2017) 422–435. doi:10.1016/j.jmsy.2017.03.010.

[6] L. Hvam, A. Haug, N.H. Mortensen, C. Thuesen, Observed benefits from product configuration systems, International Journal of Industrial Engineering: Theory, Applications and Practice. 20 (2013) 329–338.

[7] A. Trentin, E. Perin, C. Forza, Product configurator impact on product quality, International Journal of Production Economics. 135 (2012) 850–859. doi:10.1016/j.ijpe.2011.10.023.

[8] M. Heiskala, J. Tiihonen, K.S. Paloheimo, T. Soininen, Mass customization with configurable products and configurators: a review of benefits and challenges, in: Mass Customization Information Systems in Business, IGI Global, London, UK, 2007: pp. 1–32. doi:10.4018/978-1-59904-039-4.ch001.

[9] C. Forza, F. Salvador, Managing for variety in the order acquisition and fulfilment process: the contribution of product configuration systems, International Journal of Production Economics. 76 (2002) 87–98. doi:10.1016/S0925-5273(01)00157-8.

[10] M. Gronalt, M. Posset, T. Benna, Standardized configuration in the domain of hinterland container terminals, Series on Business Informatics and Application Systems Innovative Processes and Products for Mass Customization. 3 (2007) 105–120.

[11] S. Shafiee, P. Piroozfar, L. Hvam, E.R.P. Farr, G.Q. Huang, W. Pan, A. Kudsk, J.B. Rasmussen, M. Korell, Modularisation strategies in the AEC industry: a comparative analysis, Architectural Engineering and Design Management. 16 (2020) 270–292. doi:10.1080/17452007.2020.1735291.

[12] B. Squire, S. Brown, J. Readman, J. Bessant, The impact of mass customisation on manufacturing trade-offs,

Production and Operations Management. 15 (2009) 10–21. doi:10.1111/j.1937-5956.2006.tb00032.x.

[13] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, G. Petrone, R. Schafer, M. Zanker, A framework for the development of personalized, distributed web-based configuration systems, AI Magazine. 24 (2003) 93–110. doi:10.1609/aimag.v24i3.1721.

[14] G. (Jason) Liu, R. Shah, R.G. Schroeder, Linking work design to mass customization: A sociotechnical systems perspective, Decision Sciences. 37 (2006) 519–545. doi:10.1111/j.1540-5414.2006.00137.x.

[15] E. Sandrin, A. Trentin, C. Forza, Leveraging high-involvement practices to develop mass customization capability: A contingent configurational perspective, International Journal of Production Economics. 196 (2018) 335–345. doi:10.1016/j.ijpe.2017.12.005.

[16] Y. Wang, J. Wu, R. Zhang, S. Shafiee, C. Li, A “user-knowledge-product” Co-creation cyberspace model for product innovation, Complexity. 2020 (2020). doi:10.1155/2020/7190169.

[17] Y. Wang, J. Wu, L. Lin, S. Shafiee, An online community-based dynamic customisation model : the trade-off between customer satisfaction and enterprise profit, International Journal of Production Research. 0 (2019) 1–30. doi:10.1080/00207543.2019.1693649.

[18] S. Shafiee, A. Haug, S.. Kristensen, L. Hvam, Application of design thinking to product-configuration projects, Journal of Manufacturing Technology Management. (2020).

[19] K. Kristjansdottir, S. Shafiee, H. Hvam, C. Forza, N.H. Mortensen, The main challenges for manufacturing companies in implementing and utilizing configurators, Computers in Industry. 100 (2018) 196–211. doi:10.1016/j.compind.2018.05.001.

[20] S. Shafiee, Conceptual modelling for product configuration systems, Technical University of Denmark, Denmark, 2017.

[21] C. Forza, F. Salvador, Product configuration and inter-firm coordination: An innovative solution from a small manufacturing enterprise, Computers in Industry. 49 (2002) 37–46. doi:10.1016/S0166-3615(02)00057-X.

[22] A. Haug, L. Hvam, N.H. Mortensen, Definition and evaluation of product configurator development strategies, Computers in Industry. 63 (2012) 471–481. doi:10.1016/j.compind.2012.02.001.

[23] M. Heiskala, J. Tiihonen, K. Paloheimo, Mass customization of services: Benefits and challenges of configurable services, in: Frontiers of E-Business Research (FeBR 2005), Tampere, Finland, 2005: pp. 206–221.

[24] S. Shafiee, L. Hvam, M. Bonev, Scoping a product configuration project for engineer-to-order companies, International Journal of Industrial Engineering and Management. 5 (2014) 207–220.

[25] C. Forza, A. Trentin, F. Salvador, Supporting product configuration and form postponement by grouping components into kits: the case of MarelliMotori, International Journal of Mass Customisation. 1 (2006) 427–444. doi:10.1504/IJMASSC.2006.010443.

[26] T. Blecker, N. Abdelkafi, G. Kreutler, G. Friedrich, Product configuration systems: State of the art, conceptualization and extensions, in: Proceedings of the Eight Maghrebian Conference on Software Engineering (MCSEAI 2004), Tunisia, 2004: pp. 25–36.

[27] A. Felfernig, G.E. Friedrich, D. Jannach, UML as domain specific language for the construction of knowledge-based configuration systems, International Journal of Software Engineering and Knowledge Engineering. 10 (2000) 449–469. doi:10.1016/S0218-1940(00)00024-9.

[28] S. Shafiee, L. Hvam, A. Haug, M. Dam, K. Kristjansdottir, The documentation of product configuration systems: A framework and an IT solution, Advanced Engineering Informatics. 32 (2017) 163–175. doi:10.1016/j.aei.2017.02.004.

[29] A. Haug, L. Hvam, The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems, International Journal of Mass Customisation. 2 (2007) 1–18. doi:10.1504/IJMASSC.2007.012810.

[30] S. Shafiee, K. Kristjansdottir, L. Hvam, C. Forza, How to scope configuration projects and manage the knowledge they require, Journal of Knowledge Management. 22 (2018) 982–1014. doi:10.1108/JKM-01-2017-0017.

[31] D.L. Kuhn, Selecting and effectively using a computer aided software engineering tool, Westinghouse Savannah River Co., Aiken, SC (USA), 1989.

[32] Y. Zeng, K.Y. Kim, V. Raskin, B.C.M. Fung, Y. Kitamura, Modeling, extraction, and transformation of semantics in computer aided engineering systems, Advanced Engineering Informatics. 27 (2013) 1–3. doi:10.1016/j.aei.2012.12.001.

[33] L. Fowler, M. Allen, J. Armarego, J. Mackenzie, Learning styles and CASE tools in software engineering, in: 9th Annual Teaching Learning Forum, Curtin University of Technology, Perth, Australia, 2009: pp. 1–10.

[34] S. Shafiee, K. Kristjansdottir, L. Hvam, Business cases for product configuration systems, in: 7th International Conference on Mass Customization and Personalization in Central Europe, Novi Sad, Serbia, 2016.

[35] A. Haug, S. Shafiee, L. Hvam, The costs and benefits of product configuration projects in engineer-to-order companies, Computers in Industry. 105 (2019) 133–142. doi:10.1016/j.compind.2018.11.005.

[36] S. Shafiee, Y. Wautelet, L. Hvam, E. Sandrin, C. Forza, Scrum versus Rational Unified Process in facing the main challenges of product configuration systems development, The Journal of Systems & Software. 170 (2020) 110732. doi:10.1016/j.jss.2020.110732.

[37] D.C. Schmidt, Model-Driven Engineering, in: Computer-IEEE Computer Society, 2006: p. 25. http://www.computer.org/portal/site/computer/menuitem.e533b16739f5...

[38] S. Kent, Model Driven Engineering, in: International Conference on Integrated Formal Methods, Springer, Berlin, Heidelberg, 2002: pp. 286–298. doi:10.1007/978-3-319-03050-0_2.

[39] Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing. 182 (2016) 304–321. doi:10.1016/j.neucom.2015.12.022.

[40] R. Bashroush, M. Garba, R. Rabiser, I. Groher, G. Botterweck, CASE tool support for variability management in software product lines, ACM Computing Surveys (CSUR). 50 (2017) 1–45. doi:https://doi.org/10.1145/3034827.

[41] Y. Wautelet, S. Shafiee, S. Heng, Revisiting the product configuration systems development procedure for Scrum compliance: an i* driven process fragment, in: X. Franch, T. Männistö, S. Martínez-Fernández (Eds.), Product-Focused Software Process Improvement, Springer International Publishing, Cham, 2019: pp. 433–451.

[42] A.R. Hevner, S.T. March, J. Park, S. Ram, Design science in information systems research, MIS Quarterly. 28 (2004) 75–105.

[43] K.L. Cossick, T.A. Byrd, R.W. Zmud, A synthesis of research on requirements analysis and knowledge acquisition techniques, MIS Quarterly. (1992).

[44] J. van Aken, A. Chandrasekaran, J. Halman, Conducting and publishing design science research: Inaugural essay of the design science department of the Journal of Operations Management, Journal of Operations Management. 47–48 (2016) 1–8. doi:10.1016/j.jom.2016.06.004.

[45] N. Suzić, E. Sandrin, S. Suzić, C. Forza, A. Trentin, Z. Anišić, Implementation guidelines for mass customization: A researcher-oriented view, International Journal of Industrial Engineering and Management. 9 (2018) 229–243. doi:10.24867/IJIEM-2018-4-229.

[46] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, K. Wolter, Configuration knowledge representation and reasoning, Elsevier, Amsterdam, Netherlands, 2014. doi:10.1016/B978-0-12-415817-7.00006-2.

[47] J. Tiihonen, T. Männistö, A. Felfernig, Sales configurator information systems design theory, in: Configuration Workshop, 2014: pp. 67–74.

[48] A. Felfernig, D. Jannach, M. Zanker, Contextual diagrams as structuring mechanisms for designing configuration knowledge bases in UML, in: International Conference on the Unified Modeling Language, 2000.

[49] P. Kruchten, The rational unified process: An introduction, 3rd ed., Addison-Wesley Professional, 2007.

[50] A.K. Shuja, J. Krebs, IBM rational unified process reference and certification guide: solution designer (RUP), IBM Press, 2007.

[51] A. Duffy, M. Andreasen, Enhancing the evolution of design science, in: Proceedings of ICED 95, Praha, Zurich,

1995: pp. 29–35.

[52] P.Y. Chao, T. Te Chen, Analysis of assembly through product configuration, Computers in Industry. 44 (2001) 189–203. doi:10.1016/S0166-3615(00)00086-5.

[53] S. Arslan, G. Kardas, DSML4DT: A domain-specific modeling language for device tree software, Computers in Industry. 115 (2020) 103179. doi:10.1016/j.compind.2019.103179.

[54] D. Magro, P. Torasso, Decomposition strategies for configuration problems, Ai Edam. 17 (2003) 51–73. http://www.journals.cambridge.org/abstract_S0890060403171053.

[55] A. Haug, Representation of industrial knowledge as a basis for developing and maintaining product configurators, Technical University of Denmark, Denmark, 2008. http://forskningsbasen.deff.dk/Share.external?sp=S7c4c9d41-faa9-4544-bbca-fdad9b8d9970&sp=Sdtu.

[56] K. Beck, W. Cunningham, A laboratory for teaching object oriented thinking, ACM Sigplan Notices. 24 (1989) 1–6. doi:10.1145/74878.74879.

[57] J. Tiihonen, M. Heiskala, A. Anderson, T. Soininen, WeCoTin –a practical logic-based sales configurator, AI Communications. 26 (2013) 99–131. doi:10.3233/AIC-2012-0547.

[58] F. Elgh, Decision support in the quotation process of engineered-to-order products, Advanced Engineering Informatics. 26 (2012) 66–79. doi:10.1016/j.aei.2011.07.001.

[59] F. MacDonald, J. Miller, A. Brooks, M. Roper, M. Wood, Automating the software inspection process, in: Computer Aided Software Engineering, Springer, Cham, Switzerland, 1996: pp. 9–34.

[60] C. Coronel, S. Morris, P. Rob, Database systems: design, implementation, and management, Cengage Learning, Boston, MA, 2010.

[61] Y. Lirov, Computer-aided software engineering of expert systems, Expert Systems With Applications. 2 (1991) 333–343. doi:10.1016/0957-4174(91)90039-H.

[62] P. Eeles, K. Houston, An introduction to the rational unified process, in: Building J2EE Applications With the Rational Unified Process, Addison-Wesley Longman, Boston, MA, 2002: pp. 29–41.

[63] J. Nielsen, R. Molic, Ergonomic requirements for office work with visual display terminals, ISO. 9241 (1998) 11.

[64] J. Nielsen, The usability engineering life cycle, Computer. 25 (1992) 12–22.

[65] A.C. Haddix, S.M. Teutsch, P.S. Corso, Prevention effectiveness: a guide to decision analysis and economic evaluation, Oxford University Press, 2003.

[66] J.J. Phillips, Return on investment in training and performance improvement programs, Routledge, 2012.

[67] K. Kristjansdottir, S. Shafiee, L. Hvam, M. Bonev, A. Myrodia, Return on investment from the use of product configuration systems – a case study, Computers in Industry. 100 (2018) 57–69. doi:10.1016/j.compind.2018.04.003.

[68] A.B. Brown, A. Keller, J.L. Hellerstein, A model of configuration complexity and its application to a change management system Aaron, IEEE Transactions on Network and Service Management. 4 (2007) 13–27. doi:10.1109/TNSM.2007.030102.

[69] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering. 14 (2009) 131–164. doi:10.1007/s10664-008-9102-8.

[70] C. Voss, N. Tsikriktsis, M. Frohlich, Case research in operations management, International Journal of Operations & Production Management. 22 (2002) 195–219. doi:10.1108/01443570210414329.

[71] S. Shafiee, S.C. Friis, L. Lis, U. Harlou, Y. Wautelet, L. Hvam, A database administration tool to model the configuration projects, IEEE International Conference on Industrial Engineering and Engineering Management. 2019-Decem (2019) 341–345. doi:10.1109/IEEM.2018.8607654.

[72] S. Oscar, Visual Paradigm for Uml, in: International Book Market Service Limited, 2013.

[73] T. Quatrani, J. Palistrant, Visual modeling with IBM rational software architect and UML (The developerWorks

Series), IBM Press, 2006.

[74] M.A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, P. González, A CSCW Requirements Engineering CASE Tool: Development and usability evaluation, Information and Software Technology. 56 (2014) 922–949. doi:10.1016/j.infsof.2014.02.009.

[75] N. Suzić, C. Forza, A. Trentin, Z. Anišić, Implementation guidelines for mass customization: Current characteristics and suggestions for improvement, Production Planning and Control. 29 (2018) 856–871. doi:10.1080/09537287.2018.1485983.