Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Model for Quantitative Estimation of Functionality Influence on the Final Value of a Software Product

GREGOR MOLAN [1], GREGOR DOLINAR [1], JOVAN BOJKOVSKI (MEMBER, IEEE),[1], RADU PRODAN [2], ANDREA BORGHESI [3], MARTIN MOLAN [4]

[1]Faculty of Electrical Engineering, University of Ljubljana, Tržaška c. 25, Ljubljana, SI-1000, Slovenija
[2]Department of Information Technology, University of Klagenfurt, Universitätsstr. 65-67, Klagenfurt am Wörthersee, A-9020, Austria
[3]Computer Science and Engineering - DISI, University of Bologna, Mura Anteo Zamboni 7, Bologna, IT-40126, Bologna, Italy
[4]Electrical, Electronic, and Information Engineering "Guglielmo Marconi" - DEI, University of Bologna, Viale del Risorgimento 2, Bologna, IT-40126, Italy

Corresponding author: Gregor Molan (e-mail: gregor@comtrade.com).

ABSTRACT **Purpose:** The gap between software development requirements and the available resources of software developers continues to widen. This requires changes in the development and organization of software development.

Objectives: Presented is a model introducing a quantitative software development management methodology that estimates the relative importance and risk of functionality retention or abundance, which determines the final value of the software product.

Method: The final value of the software product is interpreted as a function of the requirements and functionalities, represented as a computational graph (called a software product graph). The software product graph allows the relative importance of functionalities to be estimated by calculating the corresponding partial derivatives of the value function. The risk of not implementing the functionality is estimated by reducing the final value of a product.

Validation: This model has been applied to two EU projects: CareHD and vINCI. In vINCI, the functionalities with the most significant added value to the application were developed based on the implemented model and those that brought the least value were abandoned. Optimization was not implemented in the CareHD project and proceeded as initially designed. Consequently, only 71% of the CareHD's potential value has been realized.

Conclusions: Presented model enables rational management and organization of software product development with real-time quantitative evaluation of functionalities impacts, assessment of the risks of omitting them without a significant impact. A quantitative evaluation of the impacts and risks of retention or abundance is possible based on the proposed algorithm, which is the core of the model. This model is a tool for rational organization and development of software products.

INDEX TERMS Chain derivation in graph, Computational graph, Cost estimation, Graph theory, Keras, Quantitative estimation, Software construction, Software development, Software engineering process, Time estimation.

I. INTRODUCTION

THE growth of the software industry and the increased demand for software engineers have driven the optimization of the software development process. Software development comprises two main parts: business development and product development [1], [2]. Several approaches to optimize both the business development phase (i.e., requirement definition) and the product implementation phase have already been introduced. However, questions about how to plan the software development process and translate requirements into functionalities still need to be answered. This paper presents a quantitative methodology to select the optimized requirements based on their importance while minimizing the development cost and delivering a software product with the highest end value.

The success of the software development planning step depends on the availability of domain and engineering knowledge [3]–[5]. The first crucial aspect to consider is the

best way to represent (and encode) existing domain knowledge into a form suitable to be integrated into quantitative processes [6], [7]. *Graphs* are a data structure capable of encoding complex knowledge and relationships into a form suitable for quantitative approaches [8]–[10]. Consequently, the quantitative method that is presented in this paper uses graphs to represent the structure of the software product by describing the relationship between the requirements, functionalities, and end value.

The primary assumption of this paper is that we can extract useful information by representing existing domain and engineering knowledge as a graph. The success of the proposed methodology strongly depends on the availability of this knowledge. This domain knowledge can often be readily available in organizations with well-structured product development pipelines but might be less readily available in other contexts [4]. To the best of our knowledge, this work is the first to demonstrate how well-encoded domain knowledge can be exploited to improve software development processes. Although this paper does not present a simple solution that promises to optimize all software development projects, it claims that the formalization of existing knowledge as a graph offers the possibility to (quantitatively) optimize the planning of the development process.

We distinguish between the contributions of this paper (which we call *method, model*, and *methodology*) and the cited previous results (which we call the *approach*).

The model that is presented in this paper is named the ***Mo****del for* ***q****uantitative* ***e****stimation* (*MOQE*, or *MOQE model*).

A. CONTRIBUTIONS OF THIS PAPER

The open question that either business or product development has not answered is how to find the optimized set of functionalities that satisfy planned requirements [11]. The contribution of this paper is that it introduces a quantitative methodology for finding the optimized set of functionalities that meet the requirements and achieve the highest product value. Therefore, it sits between business and product development and bridges requirement analysis and product development management. Specifically, this paper makes the following contributions:

- Creation of a quantitative model to find the optimized set of functionalities that retain the highest product value while requiring the least development resources.
- Creation of a quantitative model to determine a maximum realization of potential requirement value.
- Demonstration of the model's usefulness in two real-life projects.
- Open source implementation of the computational part of the model.

To provide a proof of concept of the proposed quantitative model, the practical application of the model is implemented for two EU projects. The first is the *vINCI*: "Clinically-*v*alidated *IN*tegrated Support for Assistive *C*are and Lifestyle *I*mprovement: the Human Link" project, which is funded by the European Union's Active Assisted Living Programme under grant agreement AAL2017-63-vINCI [12]. Here, we are focused on the development of a mobile application. The second is the CareHD: "Patient-centred Connected Health Model of *Care* for *H*untingtons *D*isease" EU-funded project, which is covered by the EXCELLENT SCIENCE - Marie Skłodowska-Curie Actions [13].

The proof of concept with implementing the MOQE model for two EU projects provides a baseline to advance using the MOQE model in software development management. Determined functionalities that retain the highest product value and, simultaneously, require the least development resources, are the measurable arguments to reduce the product development cost with the lowest reduction of the product value. Determined maximum realization of potential requirement value is a new state-of-the-art measure to validate the architecture of proposed software development.

The central part of this paper will focus on functionality analysis, as presented in Figure 1, and it is the intersection of business development and product development.

II. RELATED WORK

A. BUSINESS DEVELOPMENT

The role of business development is to identify the key challenges and opportunities for generating added value [14]. These insights are translated into crucial requirements for software products by requirement analysis [15]. Functional requirements, which are usually referred to simply as *requirements*, by definition, influence the functionalities of the end product [16].

The role of product development is to implement planned functionalities that satisfy crucial requirements [15]. Product development aims to satisfy the requirements set by business development with minimum development costs while achieving maximum product value, similar to a Minimum Viable Product (MVP) [17].

During the business development phase, the requirements and functionalities of a software product are determined at the beginning. These are used inputs for the development phase.

The development optimization of software is the proposed optimization of the connection between business and product development. Therefore, we suggest functionality analysis is a crucial phase in the software development cycle [1]. Software implementation depends on the defined requirements and desired output, the complexity of the development, the technologies used, and the available resources.

The model from 2010 is described as the formulation of an optimization model of software components selection for component-based software system (CBSS) development. The model has two objectives: maximizing the functional performance of the CBSS and maximizing the cohesion, and minimizing the coupling of software modules [18].

Some other fields besides software product development use mathematical models to model development processes.

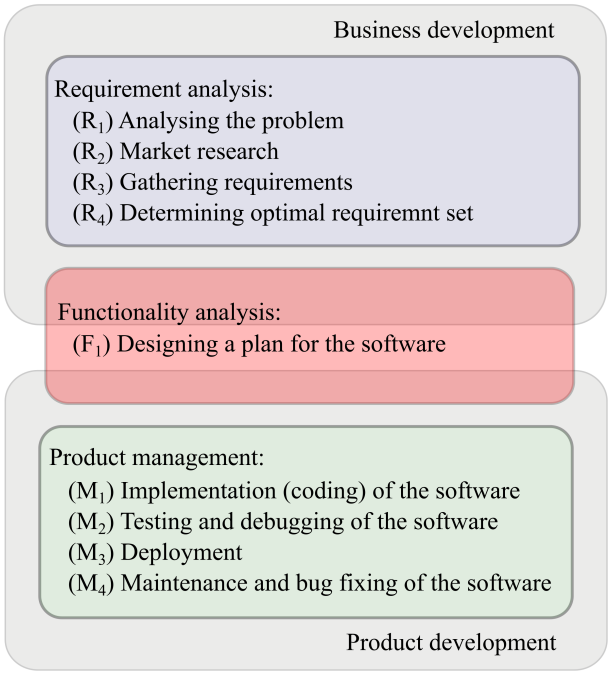

FIGURE 1. Software product development cycle.

Such a mathematical model for the development process of leather bags uses mixed-integer nonlinear programming to select the best components from available alternatives. It is modelling a make-or-buy decision on the components and selecting its suppliers [19].

To determine the value of a software product to achieve maximum product value with the given resources, we quantify collected data about software product development. This is a quantitative approach to optimizing software development. We propose the quantitative model to estimate the following items:

(a) The values of the functionalities,
(b) The influence of each functionality on the value of a software product with its gradient value,
(c) The value of a software product in the case of optimized software development.

While there are many perspectives on the value of a software product [20], studying the meaning of a term value of a software product differs from the aim of this paper. We use the term *value of a software product* to denote the final state of software product development. This paper uses the word *risk* without connection to security or vulnerability. Instead, we use risk as a term to denote the effect of not implementing the functionality and consequently reducing the value of a software product.

Recently, there have been different approaches to production modelling. Such is the stochastic model to schedule the maintenance tasks and control the inventory simultaneously in an unreliable deteriorating production system [21].

B. REQUIREMENT ANALYSIS

Requirement analysis explores the definition of the optimized set of requirements, which is the first step in the software development process. Formally, each requirement specifies a capability or a condition that must be provided by a software product [2], [22], [23].

The optimized set of requirements, as determined by requirement analysis, consists of conditions that distinguish the extent of achievement for a particular goal, contain no duplicates, and cover all business goals [24].

In recent years, requirements analysis has focused on customers. In particular, it has tried to understand their work and business needs to provide a more tailored solution that is defined by a more focused set of requirements [25]. The requirements have also been validated by quick, safe-to-fail proof of concept solutions that aim to determine the validity of a potential requirement to the customer [26].

The quality of requirement analysis also depends on the available data and approaches to take advantage of this data. Consequently, most quantitative approaches in requirement analysis are data-driven [27]. These data-driven approaches, including ML approaches, take advantage of data structure, more precisely, data ontologies [28]. Another trend in quantitative requirement analysis has combined data-driven or ML approaches with human domain experts, such as in an active learning setting [29].

The result of all requirement analysis activities is an optimized set of requirements. These requirements are fulfilled by functionalities that are implemented in the end product. Our proposed model (functionality analysis) aims to (analogous to requirement analysis) provide an optimized set of functionalities that are based on the optimized set of requirements provided by requirement analysis.

C. FUNCTIONALITY ANALYSIS

Functionality analysis is the process of evaluating the functional requirements of a system or component. It identifies the specific functions that the system or component is expected to perform and then determines how these functions will be implemented and integrated into the overall system [30], [31].

The functionality analysis aims to ensure that the system or component will meet the user's or customer's needs and requirements, which involves considering factors such as usability, reliability, performance, and maintainability [32].

Functionality analysis is typically carried out as part of the systems engineering process. It is often used to inform the design and development of new systems or to evaluate an existing system's capabilities and limitations. This is essential in developing any system because it helps identify potential problems and ensure that it meets the user's needs [33].

In software development, the main focus of functionality analysis is the translation of requirements into a set of interconnected functionalities that retain the highest product value [32].

The study of the state-of-the-art in software functionality analysis is a crucial step in our model, where we formalize the quantitative modelling of the relationship between a software product's requirements, functionalities, and value. In our model, we are extending the identification and the process of integration of determined functions into the target system.

D. PRODUCT DEVELOPMENT

Product development defines a business idea, determined by requirements, into a product ready for the end customer. According to [34], product development can be generalized to the following steps:

- (D_1) Define functionalities that address the requirements defined by requirement analysis.
- (D_2) Optimize the set of functionalities. This can be achieved by functionality analysis, the methodology proposed in this paper.
- (D_3) Implement the functionalities.
- (D_4) Deploy the final product.

Product development management aims to optimize the implementation of functionalities (in the case of software development, programming, unit testing, QA, and product documentation) to ensure that the final product will achieve the highest possible product value with the lowest cost and the lowest risk [35]. The different management approaches aim to achieve this single goal, such as Agile [36], Incremental [37], Prototyping [38], Spiral [39], V model [40], and Waterfall [41]. However, an optimized end product can only be achieved if product management approaches receive an optimized set of functionalities as input. Therefore, functionality analysis (as presented in this paper) is an essential prerequisite for any product development approach. In particular, it serves as a bridge between business development (requirement analysis) and product development.

There are many participants in the development of software products, such as business developers, project owners, project managers, systems analysts, and software developers. In 2003, the software reliability analysis model (SRAM) was developed to assist systems analysts and developers in feeling confident in predicting, measuring, ensuring, and managing the reliability of software-based systems [42].

A. FUNDATIONS OF THE MOQE MODEL

A formal definition of monitoring in the context of software engineering from the perspective of a quantitative approach started in 1989 [43]. Recently, the management process in software development organisations is also proposed as a quantitative approach [44]. There is not a single reference that requires a quantitative model for the development process, but a collection of theoretical foundations for the proposed quantitative model resumed from various perspectives described in the previous section, from business development [1], [14]–[21] , from requirement analysis [2], [22]–[29], from functionality analysis [30]–[33] , and product development perspective [34]–[42] . These approaches are the theoretical foundation of the proposed quantitative approach presented in this paper as the MOQE model.

B. THE CORE CONCEPTS OF THE MOQE MODEL

The goal of the architecture of the proposed MOQE model is to formalize the quantitative modelling of the relationship between the requirements, functionalities, and value of a software product. The MOQE consists of the following key fundamentals (basics):

b_1 : Software product graph.
The software product graph is the core of the MOQE model. It is a computational graph formally structured into the requirements partition, a subgraph of functionalities and a single value node as a particular part of functionalities. We use the term partition from graph theory as there are no connections between the nodes in the same partition. The main descriptions of the software product graph are in subsections III-D and III-E.

b_2 : Value/complexity trade-off function.
A differentiable function $f_p(x)$ with the property that $8x \quad 0 : f_p(x) \quad id(x)$; further described in subsection IV-A.

b_3 : Partial derivates.
Partial derivates of a value/complexity trade-off function for modelling the functionality importance (see subsection III-I).

b_4 : Algorithms.
Algorithms for two passes: (1) creation activation values - forward pass, and (2) value re-estimation - backward pass. A detailed explanation of algorithms is in subsections III-K and III-K.

C. ASSUMPTIONS OF THE MOQE MODEL

The quantitative MOQE model that is presented in this work is based on the following assumptions:

a_1 : The potential value of the product is defined as the sum of the value of individual requirements.
An optimized implementation, which is impossible in real life, would maintain all of the potential values of the requirements. The requirement value is maintained or lost with the specific implementation of the software product, according to the satisfaction of the requirement (or lack thereof). The percentage of potential requirement value lost depends on the implementation specifics (see the topology of the software product graph described in subsection III-E).

a_2 : The value lost due to the complexity of specific functionality is modelled by a value/complexity trade-off function.
The value/complexity trade-off function encodes the basic assumption of various software development management approaches: complex functionalities retain less value than simpler functionalities [47].

vINCI software product graph topology with 2 input partitions: “Requirements” on partition 0 and “Backend” on partition 1. The functionality subgraph consists of three functionality partitions: “Backend”, “Core” and “Edge” on partitions 1, 2, and 3. The nal “Value” partition is on partition 4.

CareHD software product graph topology with 2 input partitions: “Requirements” on partition 0 and “Backend” on partition 1. The functionality subgraph consists of ve functionality partitions: “Backend”, “Data collection” , “Action” , “Interventions” and “Edge” on partitions 1, 2, 3, 4 and 5. The nal “Value” partition is on partition 6.

a_3 : No software implementation can add to the value of the requirements.
Each requirement has a value that the software implementation aims to achieve by satisfying the requirement. In the case of the best possible implementation, the optimal value for the requirement is reached, and it is impossible to increase its value any further [47].

In relation to a_2, the value/complexity trade-off function is further de ned and described in subsection IV-A.

D. THE GOAL OF THE MOQE MODEL

The primary assumption of this model, as in assumption a_1 from subsection III-C, is that we can model the software development process as a function of its requirements and functionalities. This function is modelled as a computational graph. In our model, we name this a software product graph. The topology of this graph (i.e., the number of nodes and connections/edges) is determined by the speci c software product. Modelling a software product as a function of functionalities and requirements is the foundation that underpins the quantitative analysis of each functionality's importance (in uence) on the product value. In addition, this model allows us to estimate the impact of not-implementing speci c functionality on the end value of the product. The result of the quantitative methodology is an estimation of the in uence of the implemented or not implemented functionality on the value of the nal product.

E. THE ARCHITECTURE OF THE MOQE MODEL

Our MOQE model aims to capture the structure of software products. The software product graph is formally structured into the Requirements partition, a subgraph of functionalities and a single value node as a particular part of functionalities. All partitions correspond to the graph partitions of graph theory because there are no connections between the nodes in the same partition. Highlights of the MOQE model architecture are given in Figures 2 and 3 for the vINCI and CareHD EU projects, respectively.

F. INPUT DATA FOR THE MOQE MODEL

The input variables are initial node values for the requirement partition (R), while it is possible to check the importance of nodes in the backend functionality partition (B). The default input value for backend functionality partition nodes is 0. This means that the node is not (means not) removed from the model. The node is removed from the model with the input value for this node. The forward pass of the MOQE algorithm calculates the activation values and theoretical (maximum) realization of the model for given input values. Removing a speci c backend node by setting 1 for its input value gives the theoretical realization of the development model without this backend node.

The input values for nodes in the requirement partition (i.e., the vectors s_R) represent the relative importance of the requirements. The sum of all input values should be 1.

The architecture of the MOQE model in subsection III-E de nes a software product graph as a k-partite graph[1] with weight matrices $W^{(i;j)}$ are input values for connections between neighbouring partitions i and j.

G. VINCI SOFTWARE PRODUCT GRAPH

The function modelling of the relationship between functionalities, requirements, and value is represented as a computational graph (software product graph in our model). The computational graph for the vINCI EU projects with nodes in a software product graph is organized in partitions, as it is presented in Figures 2. The vINCI project software product graph has the following partitions, where each partition can consist of any number of nodes:

1. The requirement partition (R) represents the software product's key objectives. The values of the requirement nodes represent the value that the requirements would contribute to the end software product if it were implemented ideally. In our model, the value of the required node is maintained or reduced throughout the software product graph.
2. The backend functionality partition (B) represents functionalities supporting other functionalities but not di-

[1]Mathematically correct: It is an (n+3)-partite graph for n additional functionalities after backed functionality. We have the 5-partite graph for the vINCI and the 7-partite graph for the CareHD project example.

Partitions and nodes in vINCI software product graph

Partition	Nodes
(R) Requirements	R_1 Quality of life R_2 Cardiovascular R_3 Postures R_4 Activities
(B) Backend functionalities	B_1 Smart shoes B_2 Smartwatch B_3 Smartphone B_4 O_2 detector B_5 Cloud infrastructure
(F) Core functionalities	F_1 Fall detection F_2 Activity detection F_3 GPS monitoring F_4 Well-being detection
(E) Edge functionalities	E_1 Mobile app E_2 Smartwatch app E_3 Caregiver service
(V) Value partition	V_1 Product value

rectly addressing the requirements. Backend nodes are not connected to the requirement partition.

3. The core functionality partition (F) represents the functionalities that directly address the requirements.
4. The edge functionality partition (E) represents the functionalities that communicate the outputs of the software product to the end customer. Edge functionality nodes are directly connected to nodes from the value partition.
5. The value partition (V) represents the estimated value of the software product. The software product's value equals the (arithmetic) sum of the requirement values in an exemplary implementation that is never achievable in industry software production. In our model, which describes a real-life software product, the value partition represents the percentage of the potential requirement value that is maintained by a specific software product structure.

Short descriptions of all of the nodes in the input node, functionality nodes and the value node defined for vINCI EU project are in Table 1.

The relationships between the nodes are modeled as directed weighted graph edges (arcs). There are two kinds of arcs:

1) Weighted arcs between requirement (R) and backend functionality (B) as a source and core functionality (F) as a destination represent the percentage of a requirement value, which is realized by specific functionality.
2) Weighted arcs between core (F) and edge functionality (E) represent the dependency between the two functionalities.

Activation values $a^{(0)}$ and $a^{(1)}$ for the input partition are input variables ($a^{(0)} = r_R$ and $a^{(1)} = r_B$). In contrast, activation variables for (B) Backend functionality partition are computed by following the operations in the software product graph, as defined by the graph partitions. Input values for connections between neighbouring partitions i and j are matrices $W^{(i;j)}$. To simplify the notation, partitions (F) Core, (E) Edge, and (V) Value are defined by the following graph computations:

Core functionality partition

– Combination part is defined by the equation

$$W^{(0;2)} \; a^{(0)} \quad W^{(1;2)} \; a^{(1)} \qquad W^{(0;2)} \; a^{(0)} \tag{1}$$

– Non-negative part is defined by the following equation

$$\mathrm{id}(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{2}$$

for each x, where x is an element of the output vector of the combination part.

– Value/complexity part is defined by the equation

$$p \in [0;1] : f_p(x) = \begin{cases} 0, & \text{for } x < 0 \\ x \quad (1 \quad p)x^{\frac{1}{1\ p}}, & \text{for } x \geq 0 \end{cases} \tag{3}$$

for each x, where x is an element of the output vector of the non-negative graph part, and p is the percentage of potential requirement value. Function modelling value/complexity trade-off is further described in section IV-A. Labeling the result from equation (1) as x_c gives $a^{(2)} = f_p(x_c)$.

Edge functionality partition is defined by the equation

$$a^{(3)} = W^{(2;3)} \; a^{(2)}$$

where $a^{(2)}$ is the output vector of the core functionality partition.

The value partition is defined by the equation

$$a^{(4)} = W^{(3;4)} \; a^{(3)}$$

where $a^{(3)}$ is the output of edge partition. The result of the value partition is a scalar.

Binary operation $\quad$ in the equation (1) for Combination part denotes the Hadamard product [48] (also known as the element-wise).

H. CAREHD SOFTWARE PRODUCT GRAPH

The graph in Figure 3 gives the background for the design (architecture) of the CareHD software product graph (computational graph). The following partitions are used for the MOQE model for the CareHD EU project:

1. The requirement partition (R) is the same as in subsection III-G.
2. The backend functionality partition (B) is the same as in subsection III-G.
3. The data collection partition (D) represents the functionality that collects the requirements and backend functionality partition data.
4. The action partition (A) represents the functionality that provides actions for data collection.
5. The intervention partition (A) represents the functionality that generates interventions.

Partitions and nodes in CareHD software product graph

Partition	Nodes
(R) Requirements	R_1 Acquisition R_2 Validation R_3 Intervention
(B) Backend functionalities	B_1 Server infrastructure B_2 Smartwatch B_3 Smartphone B_4 3rd party API
(D) Data collection	D_1 Data collection
(A) Action	A_1 Storage A_2 Processing A_3 Management
(I) Interventions	I_1 Generating interventions
(E) Edge functionalities	E_1 Website E_2 Mobile app E_3 3rd party API E_4 Call support E_5 New knowledge
(V) Value partition	V_1 Product value

6. The edge functionality partition (E) represents the functionalities that communicate the outputs of the software product to the end customer. As in the vINCI model, edge functionality nodes are directly connected to nodes from the value partition.
7. The value partition (V) is the same as in subsection III-G.

Similarly, as for the vINCI EU project, short descriptions of all of the nodes in the input node, functionality nodes and the value node de ned for the CareHD EU project are in Table 2.

The relationships between the nodes are modelled as directed weighted graph edges (arcs). There are two kinds of arcs, as in the MOQE model for vINCI EU project.

I. ESTIMATING THE INFLUENCE OF THE FUNCTIONALITY

In the software product graph, the nal value of the software product is modelled as a function of the functionalities and requirements. We de ne the in uence of functionality $f^{(j)}$ in graph partition l as a partial derivative of value function v. Partial derivatives of speci c functionalities are calculated by following the chain rule for derivation $(g \circ h)' = (g' \circ h) \cdot h'$ through the software product graph.

J. ESTIMATING THE RISK OF THE FUNCTIONALITY

To estimate the risk of the functionality, the impact of not-implemented functionality quanti cation to the nal value of a product in a more quantitative way has been investigated because it was investigated and proposed as penalty rewards related to not-implemented requirements [49].

We can estimate the decrease in the nal value for backend functionalities if the functionalities were not implemented. The product's reduction to the nal value comes from the dependency between the backend and core functionalities. The in uence of the backend functionality on the core functionality is de ned as the proportional decrease of the value that the core functionality can realize. If the dependency between the backend and core functionality is weighted 1, as then core functionality cannot realize a value if the backend functionality is not implemented.

As an input to the model, value 1 represents that the backend functionality is not implemented, and 0 indicates that it is implemented. In the graph equation, the relationship between the backend and core functionalities is modelled as a term $W^{(1;2)} \cdot r_B$ in the combination part. Weighted arcs between the backend and core partitions are used in risk estimation (calculation of graph output for value 1) and importance estimation (evaluation of derivatives for value 0).

K. THEORETICAL BASIS OF ALGORITHMS

Algorithms have a central role in the presented MOQE model for quantitative estimation of the in uence of functionality on the product's nal value. The idea of the proposed MOQE model is to apply the chain rule in a generalized graph. This can be seen as a particular case of the back-propagation algorithm for neural networks [50].

The software product graph that is constructed for the product is used as the input of the MOQE model and the algorithm. The MOQE model needs to support the diverse topologies of software product graphs. Each graph partition can be connected to any partition except for the input partition, which cannot have inbound connections.

The algorithm for the MOQE model has two passes: (1) forward pass for activation value creation and (2) backward pass for value re-estimation. The result from the forward pass is a list of activation value vectors $[a^{(l)}]_{l=2}^{L}$. This output (result) from the forward pass is used as the input for the backward pass. The results from the backward pass are the functionality in uences presented as a list of vectors containing relevant gradients, shown in red rectangles at the bottom of the vertices.

Let's assume $W^{(i;j)}$ as the weighted matrix for arcs between partitions i and j. Additionally, let's assume vector function $F_p(x) = (f_p(x_1); f_p(x_2); ::::)$.

1) Forward pass algorithm

The forward pass algorithm generates the list of activation value vectors $[a^{(l)}]_{l=2}^{L}$. The high-level design of the algorithm for the forward pass is the following (the step for the partition):

S_0 Activation value vector for requirements.

$$a^{(0)} = r_R$$

S_1 Activation value vector for backend functionalities.

$$a^{(1)} = r_B$$

S_2 Activation value vector for functionalities.

$$a^{(2)} = W^{(0;2)} \cdot a^{(0)} \quad (W^{(1;2)} \cdot a^{(1)}) \quad (W^{(0;2)} \cdot a^{(0)})$$

S_3 Activation value vector for functionalities.

$$a^{(3)} = F_p(W^{(2;3)} \cdot a^{(2)})$$

$$\vdots$$

S_{L-1} Activation value vector for the edge functionalities.

$$a^{(L-1)} = F_p(W^{(L-2;L-1)} \cdot a^{(L-2)})$$

s_L The value of the model.

$$a^{(L)} = W^{(L-1;L)} \; a^{(L-1)}$$

Result: Activation value vectors $[a^{(l)}]_{l=0}^{L}$

2) Backward pass algorithm

The backward pass algorithm generates values for functionality influence. Input for this algorithm are the weighted matrix $W^{(i;j)}$ for arcs between partitions i and j, activation values $[a^{(l)}]_{l=0}^{L}$ from the forward pass algorithm, and value/complexity trade-off function f_p. Here is the high-level design of the backward pass algorithm presented with the following steps:

s_0 Activation values.

$$V(a) = [a^{(l)}]_{l=0}^{L}$$

s_1 Functionality influence for edge functionalities.

$$_E = _{L-1} = W^{(L-1;L)} \; r_E V(a)$$

s_2 Functionality influence for core functionalities.

$$_{L-2} = r_{F_p} id(W^{(L-1;L-2)}) \; _{L-1}$$

$\vdots$

s_{L-2} Functionality influence for core functionalities.

$$_2 = r_{F_p} id(W^{(3;2)}) \; _3$$

s_{L-1} Functionality influence for requirements.

$$_R = + W^{(0;2)} \; _2$$

s_L Functionality influence for backend functionalities.

$$_B = W^{(1;2)} \; _2 \quad W^{(0;2)} \; r_R$$

Result: Functionality influences: $_E; _{L-2}; :::; _2; _B; _R$

L. IMPLEMENTATION OF ALGORITHMS

Some examples of a software product graph topology are presented in Figures 2 and 3. These examples illustrate the general structure of the software product graph topology, which is the (R) Requirement partition for inputs, partitions for functionalities, and the (V) Value graph partition.

The software product graph on Figure 2 for the vINCI project has three functionality graph partitions, as follows: (B) Backend, (F) Core functionalities, and (E) Edge. The second software product graph for the CareHD project is given in Figure 3. This has five functionality graph partitions, as follows: (B) Backend, (D) Data collection, (A) Actions, (I) Interventions, and (E) Edge. The (V) Value graph partition is always placed at the end of any software product graph topology.

Forward and backward passes for the vINCI project are presented in Figures 4 and 5. All of the details about steps of algorithms for our EU projects are available in source code presented as Jupyter notebooks in Git [51].

Algorithm for creation activation values - forward pass. Arrows in Figures 2, 3, 7, and 8 present the direction of the forward pass.

A. MODELING VALUE/COMPLEXITY TRADE-OFF

The model's assumption a_2 from subsection III-C states that the functionalities that fulfil more requirements or have more dependencies retain a lower value than simpler functionalities [47]. This characteristic of decreasing value retention is modelled as a function f_p from equation (3) because the identity function represents the ideal value retention, small values are closer to identity ($\lim_{x \to 0} \frac{f_p(x)}{x} = 1$), and values approaching 1 are lower than the identity function. The parameter p describes the percentage of value that the trade-off function retains at input 1: $f_p(1) = p$. The trade-off function for both of our EU projects [12], [13] is $f_{80\%}$, which means that $p = 80\% = 0.8$.

$$f_{80\%}(x) = \begin{cases} 0, & \text{for } x < 0 \\ x - \frac{x^5}{5}, & \text{for } x \geq 0 \end{cases} \tag{4}$$

The function is required to be consistently lower or equal to the identity function for non-negative values. This models our assumption a_3 that no software implementation can add to the estimated value of the requirements [47]. Figure 6 presents the plots of the function $f_{80\%}()$ and the identity function.

The mathematical property that $\forall x \geq 0 : f_p(x) \leq id(x)$ formalizes the discovery of quantitative software approaches that increasingly complex functionalities reduce the potential value of the final software [52].

B. IMPLEMENTATION OF THE MODEL ARCHITECTURE

In this work, the estimation of the importance of the functionality is defined as a partial derivative of a value function regarding the specific functionality. These functionalities are calculated by applying the chain rule to the graph representing the topology of the software product. Specialized

Algorithm for value re-estimation - backward pass: the creation of functionality in uence. Opposite to forward pass, arrows in Figures 2, 3, 7, and 8 present the opposite direction of the forward pass.

Modeling value/complexity trade-off function $f_{80\%}(x)$ for $p = 80\%$ used for both implementations of the MOQE model in for vINCI and CareHD projects.

programming frameworks can be employed to calculate the derivatives within a software product graph. For the experimental section of this work, the software product graph is implemented with Keras [53] and TensorFlow [54]. In our specific implementation, which is presented in section V, the value function V is modelled as

$$V(a) = W^{(3;4)} \quad a^{(3)}$$

This value function can be implemented in the Keras (TensorFlow) framework as the mean squared error (MSE) loss function with true label 0. Because all values are non-negative, the squared sum function is (for calculating the derivatives) equivalent to the sum function.

A software product graph must represent a neural network to use high-level programming frameworks such as Keras. The topology of a specific network used for the experimental part is presented in Appendix A. The fulfilment of functionalities and dependencies between functionalities are represented as weights in a neural network. The mean squared errors loss function corresponds to our desired value function if we supply 0 as a training label. Information about the ideal values of the functionalities is represented as inputs to the neural network. The input values are normalized to ensure more straightforward calculations of the gradients. Derivatives that are the end product of our model are obtained from a single back-propagation pass on a described neural network. In contrast, activation values are obtained from a single forward pass.

This implementation provides a solution that quantitatively models the relationship between requirements, functionalities, and the value of a software product. The implementation in the form of the software code that implements the model presented is available in Jupyter notebooks in Git [51]. This publicly available implementation of all algorithms from this paper also contains the input data for both EU projects, vINCI [12] and CareHD [13]. Implementations and data published in the Git [51] enable the following complements to this paper:

- c_1 : Reproduction of the presented results.
  Reproduction of results from the paper is not only the additional proof of the correctness of the proposed MOQE model but also an additional view of the MOQE model that helps to understand this solution.
- c_2 : Validation of algorithms.
  The presented paper is not a formal mathematical paper with formal mathematical proofs. Insight into the practical implementation of the MOQE model in Git [51] enables the validation of presented algorithms and proves their correctness.
- c_3 : Validation of findings from the paper.
  The MOQE model implementation is, besides the validation of presented algorithms, the validation of all findings presented in the paper. The implementation shows the practical validity of the MOQE model.
- c_4 : Highlights and understanding of algorithms.
  Text about the presented algorithms cannot be sufficient for a complete and fast understanding of the solution based on the presented algorithms. The presented implementation allows another possibility for understanding algorithms with changing parts of the implementation,

changing part of input data, and investigating the results after such modi cations.

c_5 : Comparison of two implementations.
The paper presents the MOQE model that must be implemented for a given problem. Implementing the MOQE model for two different EU projects provides another insight into various possible implementations and, as such, provides a real comparison between different implementations. In this case, additional comparison of implementations for two EU projects.

c_6 : Templates to use for other cases.
The MOQE model is the model for rational management and organization of product development, not limited to software product development. The implementations for software development for two EU projects are the template for implementing the MOQE model for other industries.

c_7 : Teaching by example.
A practical example of MOQE model usage helps us understand the idea of the presented model in the way of teaching by example. This is another complimentary explanation of the presented MOQE model.

A. THE APPLICATION OF THE MOQE MODEL: VINCI

The first application of the model, which allows the development of applications with additional requirements related to the end customers and user-friendly output, was achieved for a reasonable cost. This means that the end-users can afford and use the application. The model also helps to solve additional unique challenges, such as developing applications for the growing population of older adults [12].

The development of software applications for the elderly population is adapted to this population's unique characteristics and needs. Accessible IoT devices and cloud infrastructure integration are the basis for developing accessible applications. This improves the quality of life of the elderly population with additional integrated caregiver support. Older adults are more active and independent of direct help from relatives for a reasonable cost that most end users can afford.

The development of software products is optimally organized according to the requirements and the expected outcome. The desired outcome is to integrate already developed tools and IoT devices with additional solutions in a user-friendly form for a reasonable cost. The application development process for integrating IoT information with caregiver services identifies the critical challenges in developing end-user applications. The process is optimized to cross the bridge between research ideas and the development of applications for broader use. The optimization meets the following criteria:

Use of previously developed and available devices for data collection.

Integration in cloud infrastructure to achieve practical and time-adequate caregiver support.

The developed application must be user-friendly for the target population.

Developed applications must be available for a reasonable cost and available for all devices.

The reasonable cost of the solution based on the application is the basis for massive use and consequent cost-effectiveness for the software development company.

B. THE APPLICATION OF THE MOQE MODEL: CAREHD

The second application of the MOQE model, which is the application for the CareHD project, is also calculated with the Keras (TensorFlow) framework. All of the details are available via Git [51].

C. ALGORITHMS IMPLEMENTED WITH KERAS

The algorithms presented in subsection III-K are implemented with Keras. The code that implements the model shown in section IV is available via Git [51].

The project software product graph for the vINCI project has four functionality partitions and the value node described in subsection III-G. Input variables are written in vectors r_R and r_B with values

$$r_R^I = [0.40\ 0.13\ 0.20\ 0.27] \quad \text{and} \quad r_B^I = [0\ 0\ 0\ 0\ 0]. \tag{5}$$

The proposed topology for this software product graph only has connections between neighbouring partitions. These input parameters are presented with weight matrices $W^{(ij)}$ for graph arcs between partitions i and j:

$$W^{(0;2)} = \begin{bmatrix} 0.1 & 0.2 & 0.2 & 0.5 \\ 0 & 0.4 & 0 & 0.6 \\ 0.7 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0.3 & 0.4 \end{bmatrix} \quad W^{(1;2)} = \begin{bmatrix} 0.6 & 0.4 & 0 & 0 \\ 0.6 & 0.6 & 0.8 & 0.8 \\ 0.1 & 0.1 & 0.9 & 0 \\ 0 & 0 & 0 & 0.8 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$W^{(2;3)} = \begin{bmatrix} 0.05 & 0 & 0.95 \\ 0.3 & 0.3 & 0.4 \\ 0.4 & 0 & 0.6 \\ 0.3 & 0 & 0.7 \end{bmatrix} \quad W^{(3;4)} = \begin{bmatrix} 0.8 \\ 0.5 \\ 1 \end{bmatrix}$$

The trade-off function $f_{80\%}$ from (4) is used to model the complexity value. As the methodology is described in section III-A, the proposed model aims to determine the influence of various functionalities on the final value of the software product. The functionalities, their requirements, and their relationship to the case of the project [12] are depicted in Figure 7. The influence of the functionality defined as a partial derivative, it is in Figure 7, is a partial derivative evaluated by vectors r_R and r_B. The derivates (chain derivation rule) on our directed graph in this example are calculated with Keras (TensorFlow) framework [51].

The functionalities of the project are organized in the input node, functionality nodes and the value node partitions and the value node (see subsection III-G). Core functionalities F_1 to F_4 directly address the project requirements R_1 to R_4. The core functionalities are also supported by backend functionalities B_1 to B_5. The information generated by core functionalities is communicated to the customer by edge

Software product graph for the vINCI project with ve partitions. Vertices in the requirement partition are R , and in functionality partitions are B , F , and E . Arrows in this graph are directed in the forward pass of the algorithm for the MOQE model (it is the algorithm for creation activation values) while the backward pass (it is the algorithm for value re-estimation) proceeds in the opposite direction (for the algorithm see subsection III-K.

functionalities E_1 to E_3. Weighted directed connections between functionalities (graph arcs) represent the dependency strength between the functionalities. In contrast, the arcs between requirements and core functionalities represent the percentage of requirement value realized by speci c functionality.

As presented in Figure 9, backend functionalities are in uenced by functionality importance and functionality risk. The details of the Functionality importance and Functionality risk are as follows.

To estimate the importance of the functionality, partial derivatives of the value function are evaluated in vectors $\mathbf{0}_B$ and $\mathbf{r}_B$, as given in equation (5). Importance is calculated as an absolute value of the partial derivatives. According to the results presented in Figure 7, a cloud infrastructure B_5 is the most vital backend functionality ($\frac{\partial V}{\partial B_5}(\mathbf{0}_B) = 4{:}41$). Cloud infrastructure supports all of the core functionalities necessary for the system to function. The second most vital backend functionality is a smartwatch B_2. A smartwatch is a data source that can support all other core functionalities. Compared to more specialized functionalities, such as smart shoes B_1 or oxygen sensor B_4, it is less ef cient in helping each speci c functionality but is universal. Based on the presented results, it would be wise when optimizing backend functionalities development to focus more resources on the development of the essential functionalities, which are cloud infrastructure and smartwatch integration.

All of the core functionalities are roughly equally important. Based on the value of our requirements, fall detection F_1 and well-being detection F_4 have a slightly more substantial in uence on the product's nal value.

The most crucial edge functionality is the caregiver service E_3. This is given by the EU vINCI project's assumptions, which prioritize the development of an application that can automatically alert caregivers if their ward is in danger (e.g., fall or heart problems). Mobile application E_1 is the second most crucial edge application because it is intended to keep the relatives of the elderly informed. The least essential edge functionality is the smartwatch app E_2 because it is only used to display some information (which is not critical) to the elderly user.

D. FUNCTIONALITY RISK

Functionality risk can be estimated for backend functionalities. Functionality risk is an estimated decrease in software product value if a certain backend functionality is not implemented. We have two importance measures for backend functionalities: their relative importance if implemented (partial derivative evaluated in value 0) and product value decrease (decrease of the value function for value 1). The risk assessment for backend functionalities is presented in Table 3 presents the risk assessment for backend functionalities.

As expected, the absence of cloud functionality reduces the product's nal value to 0. Given that all of the core functionalities critically depend on the cloud infrastructure, no requirement value can be realized without cloud functionality. The most crucial backend functionality is the smartwatch because it is the most important data source for all core functionalities. Cloud infrastructure and smartwatches are also estimated to be the most in uential functionalities when implemented.

The least impact on the nal value is the absence of the smartphone, which reduces the value of the end product to 73%. Interestingly, the smartphone is not the least in uential functionality when implemented. The least in uential functionality when implemented is O_2 with $\frac{\partial V}{\partial B_4}(\mathbf{0}_B) = 0{:}77$, as presented in Figure 7. However, it has a greater impact on the nal value, reducing it to 61%. From the development management position, the O_2 sensor must be implemented at last. Despite not being very important (when implemented), it can signi cantly impact the product's nal value if it is omitted. Similar risks from functionalities are present even with different parameters for value/complexity trade-off function on the abscissa in Figure 10. The trade-off function is explained in subsection IV-A.

Functionality risk assessment is another tool that can be used to assess the value and importance of backend functionalities. By examining both the importance and risk, we can get a clearer insight into how to channel and prioritize functionalities in the development process.

Software product graph for the CareHD project with seven partitions. Vertices in the requirement partition (partition 0) are R and in functionality partitions (partitions 2-5) are B , D , A , I and E . The nal value is on partition 6. The same as in Figure 7, arrows in this graph are directed in the forward pass while the backward pass proceeds in the opposite direction (for the algorithm, see subsection III-K.

Resource distribution among the vINCI partners

Participant	Total EM	Participant	Total EM
1. ICI	29.5	6. AUT	26.0
2. MPU	29.5	7. SAL	27.0
3. UNRF	22.5	8. NIGG	12.5
4. NIT	21.5	9. OPL	31.0
5. CMD	8.5	10. CTR	36.0

In uence on backend functionality.

The impact of not-implementing backend functionalities (vINCI project)

Retained	Status of backend functionalities
90%	Implemented all
73%	Without smartphone
71%	Without smart shoes
61%	Without O_2
26%	Without smartwatch
00%	Without cloud infrastructure

According to the given input data for requirements and all arcs weights in the development graph, we get 90% realization of potential requirement value.

E. IMPLEMENTATION OF THE MOQE MODEL IN THE DEVELOPMENT OF AN APPLICATION FOR THE ELDERLY POPULATION

1) Determination of resources according to de ned requirements

The resource distribution among the partners at the beginning of the project is presented in Table 4.

2) Implementation of the MOQE model

Due to limited resources, the remaining production partner had to prioritize the development of certain functionalities with the Implementation of the presented model. According to this Implementation, cloud infrastructure is the most crucial backend functionality, and the second most important is the smartwatch.

(a) The cloud is more than ve times more important than the least essential functionality,
(b) The smartwatch is more than four times more important than the least necessary functionality,
(c) The cloud is about 40% more critical than the smartwatch functionality.

Due to limited resources, the remaining production partner focused primarily on the mobile application, including smart-

Total score values for vINCI are presented on the left-hand ordinate. Additional resources, which are presented on the right-hand ordinate, for backend functionalities are 5 EM for B_1, 4 EM for B_2, and 2 EM for each of B_3, B_4 and B_1.

Impact of not-implementing backend functionalities given different value/complexity trade-offs [51] for vINCI (upper) and CareHD (lower)

watch handling and the cloud. It was estimated that another smartwatch, which would provide all of the required information, had to be introduced. According to the model, the remaining production partner prioritized the implementation of smartwatch functionalities and decided it would be better to introduce the other type of watch.

As described in the section III-E The architecture of the MOQE model, backend functionalities depend on Functionality Importance and Functionality Risk, as described in subsection V-D. The resignation of one of the partners is covered with additional resources that are appropriately distributed to backend functionalities according to their importance and risks. Cloud functionality B_5 has the highest sum of importance and risk. The total score of importance and risk for cloud functionality B_5 is 2. Additional development needs are estimated from the sum of importance and risks, as presented in Figure 11.

Overall, the estimated new resources for additional development to cover the resignation of one of the partners has been an additional $(5 + 4 + 2 + 2 + 2)$ EM $= 15$ EM .

The model for software product development evaluation.

3) Estimated investment for the remaining production partner

Development of the adopted mobile application demands an additional investment of 15 EM . The management agreed with this solution if the production partner got exclusive productization rights.

F. A GENERALIZATION OF THE MOQE MODEL

The proposed and developed model is generalizable for implementation in the planning of the software development process.

The model, visually presented in Figure 12 as an extension and accomplishment of software product development highlights shown in Figure 1, comprises seven steps organized into three groups. Group (a) includes the first two steps for business development, group (b) includes four steps for software development, and in the final group (c) product development is the final step for creating the final software

Resource distribution among the CareHD partners. Partner UU is participating in the project without nancing from the EU.

Participant	Total EM	Participant	Total EM
1. UCD	29.3	6. UR	11.4
2. DAI	8.2	7. CT	6.5
3. SHAI	4.9	8. BHI	6.5
4. ICI	4.9	9. ICF	8.7
5. UA	14.7	10. ORH	4.9

product:

(a) Business development steps
   1. De nition of requirements
   2. De nition of the product goal

(b) Software development steps
   3. Creation of software product graph
   4. Implementation of
      Algorithm 1. Activation value creation
      Algorithm 2. Value re-estimation
   5. Identi ed functionality importance
   6. Identi ed functionality risk

(c) Product development step
   7. The nal decision on the software product

The nal goal of the model is to optimize the software process with particular concern for backend functionalities and to identify their importance to the nal software product. The presented model integrates graph theory to formalize and identify the relationships between functionalities, their power of in uence, and the developed model's relative importance of each functionality. The relative importance of the functionality is estimated by applying chain rule derivation to a software product graph.

G. THE SECOND APPLICATION OF THE MOQE MODEL: CAREHD

The second application is based on the available requirements [13] and created by the software product graph presented in Figure 8. This graph determined two critical bottlenecks in project management: data collection ( ) and gathering interventions ( ). According to the given input data for the requirements and other input data for the graph construction in Figure 8, we achieved 71% of maximum realization of potential requirement value.

Similar to the beginning of the vINCI EU project, the resource distribution among the partners at the beginning of the CareHD project is presented in Table 5.

Some problems and con icts among partners occurred during the project. Consequently, one of the two production partners withdrew from the project after spending the allocated budget. As a result, the remaining partner was asked to take over production.

Detailed interpretation of the results for this project is possible using the results presented on the publicly available Jupyter notebook [51], the data in gures in Appendix A and in Figures 8 and 10 related to the CareHD project, and the

Maximum realization of potential requirement value

Maximum realization	Project
71%	CareHD
90%	vINCI

interpretation for the vINCI project presented in the previous subsection.

H. RESULTS FOR DECISION-MAKERS

The most important advantage of the quantitative results from the MOQE model are values that help decision-makers to validate different solution approaches for their projects. The signi cance of the ndings from vINCI and CareHD projects are highlighted in all previous subsections, especially in V-A, V-B, V-E, and V-G. Quantitative insight into the signi cance of the ndings from EU project's implementations are presented in Figures 9 and Table 6. Available insight into the source code available via Git [51] provides another detailed insight into these implementations' ndings.

Besides results about the risk assessment for backend functionalities presented with values in Table 3 and Table 6, some values represent in uences' quantitative values (gradients) presented in Figure 7 and Figure 8 in red rectangles at the bottom of the vertices. All these values present rigorous results that help decision-makers to quantify their decisions.

I. PRACTICAL IMPLEMENTATION CHALLENGES

The main challenge of the practical implementation of the model was the formal de nition of input data, practical implementation of the algorithms presented in Figures 4 and 5, and presentation of nal results. Based on the ndings of the MOQE, the steering committee of the vINCI project decided to modify the project. Jupyter notebook, available as an open-source solution on Git [51], is our attempt to solve all these issues.

J. IMPLICATIONS FOR INDUSTRY

The presented MOQE model is developed as a quantitative model to provide the highest value of the product while requiring the least development resources. It is an ethical approach to software development and provides a solution for existing industries and start-ups. It is extremely useful to provide the highest value of the product for available resources, not to force the existing resources to provide unrealistic results. Especially for start-ups, using the MOQE model as the tool for managing product development will drastically increase the possibility of the company's success.

K. ETHICAL CONSIDERATIONS

The classical goal of software product development is pro t. The proposed MOQE model provides the ethical solution to get a pro t with the assumption of xed development resources and requirements retaining to maximise the product value. The highest product value means the highest pro t of

a product, and xed development resources mean avoiding extra pressure on developers.

The proposed MOQE model provides a new ethical solution as a tool to avoid extra pressure on developers as we assume the xed development resources.

This paper presents a proof of concept for the proposed MOQE model, demonstrating two EU projects: vINCI [12] and CareHD [13]. The presented model is based on quantitative approaches because the model's inputs were the software product requirements; proposed backend, core, and edge functionalities; and the connections between the collected functionalities.

A. IMPLEMENTATIONS OF THE MOQE MODEL

Implementing the MOQE model suggests the nal decision of tailoring the vINCI project activities. The consortium had to modify the nal productization, and they had to focus on achievable possible outcomes. The MOQE model was helpful as a tool for decision-making on the management level and modifying development activities.

The second application of the MOQE model in the CareHD EU project identi ed two project management bottlenecks: (1) data collection and (2) gathering interventions. Based on the input data, the model determined 71% maximum realization of potential requirement value. The results identi ed the need for the project consortium to consider relocating resources. However, the optimization step of the MOQE model was not executed.

B. HELP FOR DECISION MAKERS

The presented model for functionality analysis offers possibilities for analyzing and optimizing the software development process and supporting management decisions. The foundations of the model are in-the-project-determined relations and the importance of all functionalities. These relations were also the foundation for the division of resources within the project. This model also serves to support adaptation during the development process.

Methodologies without quantitative measures provide descriptive and literal quanti cations of their improvements. Such results of non-quanti er methods are useful for presentations for managers and marketing, where just descriptive comparisons between different methods are enough.

In an actual situation, the developed generalized methodology is a valuable tool in the design and development phase. It is also a helpful tool for management decisions.

In practice, however, decision-makers and marketing require clear and measurable comparisons between different solutions or different proposals for solutions. Such requirements are solved only by quantitative methods. In addition to deciding whether a certain solution is better or worse than another, quantitative methods offer an answer about the amount of difference between the methods, e.g. the solution obtained by method A is 3.4 times better than the solution by method B.

The impacts of a project often in uence the optimization of the software development process. This requires realistically estimating the necessary resources for developing the product or application. This can lead to the development of a product without all of the promised functionalities that are missing requirements, has inadequate quality, and ultimately leads to misspent development resources. As illustrated in a real-life product as part of two EU projects, the model presented in this work offers a quantitative framework to help better distribute development funds, resulting in a software product with a higher-end value.

The MOQE model is an upgrade of the existing approaches integrated with graph theory application. It offers a tool for optimizing the missing and less optimized software production process. It also provides the possibility to optimize the functionality of the production process.

Although our modelling is inductive, the nal result is a general model. Therefore, the results of this work are applicable in the business development process and the organization of resources for software development.

C. PROPOSED FUTURE IMPROVEMENTS

This paper, with the publicly available solutions of the MOQE model implemented for 2 EU projects [51], allows other researchers of the management and organization of software product development to implement the MOQE model solution for their use cases.

Moreover, the presented publicly available solution allows the implementation of the MOQE model for product development in other areas not limited to software products. Wide use of the MOQE model will provide new open questions, the background for future research work and new research questions. Finally, many new implementations of the current MOQE model will provide extensions and improvements to the current MOQE model version presented in this paper.

Topology in the vINCI project:

Topology in the CareHD project:

.

[1] IEEE. Ieee/iso/iec 29148-2018 - iso/iec/ieee international standard - systems and software engineering – life cycle processes – requirements engineering. IEEE Std 830-1984, 11 2018.

[2] S. Shatz. Towards complexity metrics for ada tasking. IEEE Transactions on Software Engineering, 26(08):1122–1127, aug 1988.

[3] L Baum, M Becker, L Geyer, A Gilbert, G Molter, and V Tamara. Supporting component-based software development using domain knowledge. In Proc. of 4th IIIS World Multiconference on Systemics, Cybernetics and Informatics (SCI2000), Orlando, USA. Citeseer, 2000.

[4] Ali Niknafs and Daniel M Berry. The impact of domain knowledge on the effectiveness of requirements idea generation during requirements elicitation. In 2012 20th IEEE International Requirements Engineering Conference (RE), pages 181–190. IEEE, 2012.

[5] Carine Khalil and Sabine Khalil. Exploring knowledge management in agile software development organizations. International Entrepreneurship and Management Journal, 16(2):555–569, 2020.

[6] Eran Rubin. Domain knowledge representation in information systems. PhD thesis, University of British Columbia, 2009.

[7] Senthil K Chandrasegaran, Karthik Ramani, Ram D Sriram, Imré Horváth, Alain Bernard, Ramy F Harik, and Wei Gao. The evolution, challenges, and future of knowledge representation in product design systems. Computer-aided design, 45(2):204–228, 2013.

[8] Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. Graph Representation, pages 259–269. Springer International Publishing, Cham, 2019.

[9] Jinjiao Lin, Yanze Zhao, Weiyuan Huang, Chunfang Liu, and Haitao Pu. Domain knowledge graph-based research progress of knowledge representation. Neural Computing and Applications, 33(2):681–690, 2021.

[10] Peng Cui, Lingfei Wu, Jian Pei, Liang Zhao, and Xiao Wang. Graph representation learning. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, Graph Neural Networks: Foundations, Frontiers, and Applications, pages 17–26. Springer Nature Singapore, Singapore, 2022.

[11] George Kousiouris. and Dimosthenis Kyriazis. Functionalities, challenges and enablers for a generalized faas based architecture as the realizer of cloud/edge continuum interplay. In Proceedings of the 11th International Conference on Cloud Computing and Services Science - CLOSER,, pages 199–206. INSTICC, SciTePress, 2021.

[12] vINCI Project Consortium. Clinically-validated integrated support for assistive care and lifestyle improvement: the human link, 2018. [Online; accessed 29-April-2021].

[13] CareHD Project Consortium. Carehd - patient centered connected health model of care for huntingtons disease, 2023. [Online; accessed 08-January-2023].

[14] Antoine Burret. The added value of co-working spaces in the area of business development support: Encouraging peer networks. Revue de lEntrepreneuriat, 13(1):51–73, 2014.

[15] Senay Tuna Demirel and Resul Das. Software requirement analysis: Research challenges and technical approaches. In 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pages 1–6. IEEE, 2018.

[16] Space Science Library and D.D. Defense. Systems Engineering Fundamentals. Supplementary Text Prepared By The Defense Acquisition University Press Fort Belvoir, Virginia. CreateSpace Independent Publishing Platform, 2016.

[17] Valentina Lenarduzzi and Davide Taibi. MVP explained: A systematic mapping study on the de nitions of minimal viable product. In 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 112–119, 2016.

[18] C.K. Kwong, L.F. Mu, J.F. Tang, and X.G. Luo. Optimization of software components selection for component-based software system development. Computers & Industrial Engineering, 58(4):618–624, 2010.

[19] Muhammad Adha Ilhami, Subagyo, and Nur Aini Masruroh. A mathematical model at the detailed design phase in the 3dce new product development. Computers & Industrial Engineering, 146:106617, 2020.

[20] Jelle de Groot, Ariadi Nugroho, Thomas Bäck, and Joost Visser. What is the value of your software? In 2012 Third International Workshop on Managing Technical Debt (MTD), pages 37–44, 2012.

[21] Seyed Mohammad Hadian, Hiwa Farughi, and Hasan Rasay. Development of a simulation-based optimization approach to integrate the decisions of maintenance planning and safety stock determination in deteriorating manufacturing systems. Computers & Industrial Engineering, 178:109132, 2023.

[22] Philippe Desfray and Gilbert Raymond. Chapter 7 - models for phase a: Vision. In Philippe Desfray and Gilbert Raymond, editors, Modeling Enterprise Architecture with TOGAF, The MK/OMG Press, pages 103–133. Morgan Kaufmann, Boston, 2014.

[23] R. Radhakrishnan, H. Carter, P. Alexander, P. Wilsey, and P. Frey. A formal speci cation and veri cation framework for time warp-based parallel simulation. IEEE Transactions on Software Engineering, 27(01):58–78, jan 2002.

[24] Lorenzo Fiorineschi, Niccolò Becattini, Yuri Borgianni, and Federico Rotini. Testing a new structured tool for supporting requirements' formulation and decomposition. Applied Sciences, 10(9):3259, 2020.

[25] Rachida Hassani and Y. Idrissi. Normalization of requirements speci cation document on software project management. J. Softw., 13:232–241, 2018.

[26] A. Bennaceur, A. Zisman, C. McCormick, D. Barthaud, and B. Nuseibeh. Won't take no for an answer: Resource-driven requirements adaptation. 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pages 77–88, 2019.

[27] Z. Zhang, Lin Gong, Y. Jin, J. Xie, and J. Hao. A quantitative approach to design alternative evaluation based on data-driven performance prediction. Adv. Eng. Informatics, 32:52–65, 2017.

[28] Antonio A. Lopez-Lorca, Ghassan Beydoun, Rafael Valencia-Garcia, and Rodrigo Martinez-Bejar. Supporting agent oriented requirement analysis with ontologies. International Journal of Human-Computer Studies, 87:20–37, 2016.

[29] Myron Hecht, Jaron Chen, and Phanitta Chomsinsap. Claim: An enhanced machine learning technique for discrepancy report analysis. In 2020 Annual Reliability and Maintainability Symposium (RAMS), pages 1–7, 2020.

[30] ISO/IEC/IEEE. Iso/iec/ieee international standard - systems and software engineering–vocabulary. ISO/IEC/IEEE 24765:2017(E), pages 1–541, 2017.

[31] IEEE. Ieee guide for developing system requirements speci cations. IEEE Std 1233, 1998 Edition, pages 1–36, 1998.

[32] Charles Wasson. Textbook - System Analysis, Design, and Development: Concepts, Principles, and Practices - 1st Edition. Wiley-Interscience, 12 2005.

[33] Jean-Luc Voirin. Model-based system and architecture engineering with the arcadia method. In Jean-Luc Voirin, editor, Model-Based System and Architecture Engineering with the Arcadia Method, pages 353–355. Elsevier, 2018.

[34] James Marquis and Ruba S. Deeb. Roadmap to a successful product development. IEEE Engineering Management Review, 46(4):51–58, 2018.

[35] C. K. Riemenschneider, B. C. Hardgrave, and F. D. Davis. Explaining software developer acceptance of methodologies: A comparison of ve theoretical models. IEEE Transactions on Software Engineering, 28(12):1135–1145, dec 2002.

[36] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. [Online; accessed 17-July-2022].

[37] C. Larman and V.R. Basili. Iterative and incremental developments. a brief history. Computer, 36(6):47–56, 2003.

[38] A.M. Davis. Operational prototyping: a new development approach. IEEE Software, 9(5):70–78, 1992.

[39] Barry Boehm. A spiral model of software development and enhancement. SIGSOFT Softw. Eng. Notes, 11(4):14–24, 1986. 12948.

[40] Kevin Forsberg and Harold Mooz. The relationship of system engineering to the project cycle. INCOSE International Symposium, 1(1):57–65, 1991.

[41] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In Proceedings of the 9th International Conference on Software Engineering, ICSE '87, page 328–338, Washington, DC, USA, 1987. IEEE Computer Society Press.

[42] Bahador Ghahramani. Software reliability analysis: a systems development model. Computers & Industrial Engineering, 45(2):295–305, 2003. 25th International Conference on Computers and Industrial Engineering.

[43] Barbara A. Kitchenham and John G. Walker. A quantitative approach to monitoring software development. Softw. Eng. J., 4(1), jan 1989.

[44] Carlos Ardila, Francisco Pino, and CÃ©sar Calvache. Fqmap: Towards a framework quantitative management of processes in small software development organizations. Periodicals of Engineering and Natural Sciences (PEN), 11:69–93, 07 2023.