TEM: A Transparency Engineering Methodology Enabling Users' Trust Judgement

Baraa Zieni
dept. Computer science
University of Leicester
Leicester, UK
Bz60@leicester.ac.uk

Reiko Heckel
dept. Computer science
University of Leicester
Leicester, UK
rh122@leicester.ac.uk

***Abstract*— Transparency is key to enhancing users' trust by enabling their judgment on the outcomes and consequences of a system's operations. This paper presents the transparency engineering methodology (TEM) to generate transparency requirements that enable users' trust judgement. The idea is to identify where transparency is lacking and to address this through patterns augmenting the specification of data, use case, and process requirements. Due to the complexity of software, it is impossible (and undesirable) to achieve full transparency throughout the system. However, transparency can be improved for selected system aspects. This is demonstrated using the results from an industrial case study with a medical technology company where, with the help of TEM, existing functional requirements were refined, and transparency requirements generated systematically.**

Keywords—Transparency, Trust, Trust Judgement, Software Engineering, Requirements Patterns, GDPR

I. INTRODUCTION

Trust is a relationship between a trustor and a trustee [1], in our case an end users' relationship with a software system. Trust related to the end user includes both initial and ongoing trust [2]. Previous research stated that trust can be systematically engineered into a system [3], e.g. using software patterns [4] and factor analysis [5] to investigate the concept of trust in software.

For a user to have trust in a system they must know about the system's qualities, functionality, and data to make an informed decision (trust judgement) on how to use the system for achieving their goals. Improving trust judgement can lead to an increase or decrease in trust, but in either case this will benefit users, enabling them to decide the utility of the system in fulfilling their goals [6]. This will prevent them from investing time and effort in an unsuitable system [7]. Trust judgment is defined as the user's ability to assess a system property of concern based on their needs and expectations. It includes an assessment of trust degree and a decision to trust or distrust the trustee [8]. This judgement occurs during both the initial and the ongoing interaction of the user with the system. To enable trust judgement, users need to receive appropriate information about the system when they need it and at the right level of detail.

To provide this information, software needs to be built with the deliberate aim of being transparent [9]. A system is transparent to a user if it provides relevant information about its functionality and data. The user requires this information for a better experience with the system, as well as to enable their trust judgment. "Good transparency […] enables people to make informed decisions" [10]. Thus, if a system is trustworthy, i.e., objectively able to satisfies the users' needs, transparency enhances trust by helping users achieve their goals. Transparency has been defined as a "user-centric principle" as it refers to openness and disclosure of information to the user, e.g., in compliance with data protection law, business and governance [11]. Transparency is also a fundamental principle for data processing under General Data Protection Regulation [12].

Existing research examined transparency as a non-functional requirement (NFR) in its interrelations with other NFRs and goals [13, 14]. However, there is a lack of understanding of transparency in relation to functional requirements. Therefore, our engineering methodology addresses transparency requirements on the system's data and functionality [15, 16]. This can increase trust in three ways:

1. Building initial trust by providing information about the functionality and limitations of the system [2, 17].
2. Maintaining ongoing trust by helping users to perform tasks more effectively while interacting with the system [1, 2].
3. Improving trust judgment, leading to more informed decisions, and managing risks while using the system [18-20].

Thus, transparency engineering can result in a more transparent system enhancing users' trust [14, 15]. The aim of this paper is to present the methodology to generate transparency requirements for functional goals from use cases or user stories. It also includes the data the system holds, how and where data is stored and who has access to view and process it etc. [21, 22]. Consequently, use case, data, and process transparency requirements are derived to ensure that this information is communicated to the user.

The remainder of the paper is organized as follows. We provide an overview of background and related work, including definitions of trust and transparency in Section II. After a description of the transparency engineering methodology in Section III, we present transparency requirements patterns. This

is followed by a presentation of a case study used for evaluation in Section IV, and the discussion of its results, conclusion, and future work in Section V.

II. Background and Related work

In this section we review the literature on transparency and its relation to trust including the engineering challenges this pose. The notion of transparency appears in a variety of domains. Turilli and Floridi [23] describe transparency as the possibility of information, intentions or behaviours to be accessed through a process of disclosure. Similarly, it is considered an important concept that can support users in their decision making. Transparency is said to be dependent on the context it is used in and can thus be referred to as a quality-in-use. According to Bevan and Azuma [24], quality-in-use in ISO/IEC DIS 14598 can be measured in terms of how effectively and efficiently users can attain their required goals in a specific environment. This helps in operationalising the notion of transparency in our research.

Cysneiros et al. [25] consider transparency as an important requirement needed for new technology and more robust systems to be adopted by users. They also state that NFRs like trust and privacy are equally important. Likewise, transparency is examined as a non-functional requirement in Marcio et al. [9] and Hosseini et al. [26]. Hosseini states that achieving transparency is hard and Marcio et al. believe it "rarely can be satisfied" and can only be accepted within limits. The researchers [9] in use i* to model the transparency of a company's actions. Their work focusses on understanding the relationship between transparency and other system qualities like security, trust, and privacy. The research used Soft-goal Interdependency Graphs (SIG) to capture these relationships, then an ontology to collect information about transparency and other non-functional requirements. Similarly, the work examined the relationship between trust and transparency and found that their correlation can be either positive or negative. Their view on software transparency is that it can range between low and high on a continuous scale based on what the system delivers. However, our research describes transparency as analogous to correctness; a system is either transparent if it provides adequate information to the user, or it is not. We focus on data, use case, and process transparency while other work was limited to information transparency only.

Focussing on privacy in the context of GDPR, a systematic approach to transparency during the development process is presented by Meis et al. [27]. Their research investigated the flow of personal data in a system to generate the privacy and transparency requirements. Those requirements are static in the sense that they help the user to understand what data the system holds on them, but not how it may be changed, or how to execute their rights under GDPR. We develop a set of static and dynamic patterns that generate transparency requirements about user data and system functions. These patterns help to inform the user about their data rights under GDPR, who has access to their data, how it has been stored, and about system functions (use cases and processes) with their alternative sequences at runtime. Further, we focus on the application of these patterns in agile development.

Trust is defined as the imperfect judgement of the likelihood that individual users' needs, and expectations will be met [28]. One way to focus on users' trust is by looking at the decisions they need to be made based on their subjective assessment of risks and benefits. Here, deficient information will lead to uncertainty in the (expected) outcomes [29]. Trust therefore is now becoming a more crucial issue especially when it comes to stakeholders making decisions with the software. Based on our analysis, we adopt the following definitions and terminology. *Trust* is a relationship formed by the interaction between the entities involved. Trust is dynamic, just as human relationships are. That means a trust relationship is subject to change based on feedback from these interactions whose results are evaluated by the participants and, where considered relevant, contribute to developing or eroding the relationship. Trust is cumulative; one single result from an interaction may not have a significant impact on the relationship, but, a repetition of similar results are most likely to have a combined effect [1].

Hoffmann et al. [4] stated that users often adopt trust to help reduce complexity and simplify the interaction with software. Trust is not directly tangible and must be analysed through collecting the antecedents of trust. Trust antecedents are "factors or elements that build trust" [4]. The researchers collected antecedents derived from existing studies and designed patterns to address them creating high-level goals one of which is transparency. The aim is to "help requirements analysts to address trust on a basic level". However, they concluded that more research is required to achieve this at a more detailed level of requirements [4]. Our work provides such a detailed approach, using TEM to refine high-level goals by extending functional requirements.

Transparency has several facets [26] and can carry several inconsistent meanings [30]. The definition of transparency in the context of this research is analogous to correctness: Software is transparent or not. It cannot be considered over-transparent or under-transparent. Transparency means to provide users with adequate information for informed choices whether to trust a system, when and how to use it to achieve their goals [31]. In this research, we adopt the view that transparency leads to more informed decisions and to user trust. Hence, we argue it is important to empower the end user by giving them control over their data (i.e., being informed of how their data is used and about their rights), and knowledge on their actions with the system. This information, to be more relevant, understandable, and actionable, needs to be revealed in relation to the users' goals within the system. We advocate that transparency can be engineered systematically during the development process. Transparency requires the appropriate of information the user needs from the system to use it successfully and exercise informed trust judgment, e.g, not leaving them to trust blindly in how systems handle their data [32]. For example, a user may give consent to use their personal data if they understand why it is needed and how it is going to be used. The appropriate of information can be provided by

hiding as well as disclosing information, i.e., it can be more or less than currently presented in a system that is not transparent. The motivation for not referring to an over-supply of information as transparent should be obvious: too much information can hide the essentials needed to make informed decisions.

Transparency is seen as a meta-requirement [24, 33]. i.e., it addresses "how requirements can be fulfilled" [33]. As [34] discusses, a transparent system must provide information relevant to the user's concern. Additionally, [35] emphasise that the information needs to be relevant to the user's goals as well as their personal data to elecit a high level of trust. Hence, if software is trustworthy (objectively adequate to the stakeholder's needs) transparency can be used to enhance the stakeholder-system relationship by inducing trust [14, 15, 35].

III. The Transparency Engineering Methodology

In this section we introduce our Transparency Engineering Methodology TEM, including patterns to create transparency requirements and their application. The approach of TEM is to identify where a system lacks transparency about its data, use cases, and processes and address these shortcomings using transparency engineering patterns. The methodology focuses on users' goals to generate requirements. This supports user empowerment by keeping their concerns at the centre of the methodology and improves clarity in the methodology's presentation. Goals of other stakeholders, such as the owners of the system or developers, are not considered directly when generating transparency requirements, but later during a consistency check which ensures that no information is revealed that might cause harm to these stakeholders, e.g., through data or security breaches, or the revelation of commercial secrets.

The design process of TEM followed a design-science approach as applied to guiding the creation of engineering artefacts [36] and Information Systems [37]. This approach is based on six steps: 1. problem identification; 2. definition of objectives; 3. design and development;4. demonstration; 5. evaluation; and 6. communication. We briefly explain how we follow steps two and three. The first step was conducted as a literature review, summarised in Section II, which identified and justified the problem. We illustrate the last step, communicating the artefact developed (TEM), throughout this paper and cover demonstration and evaluation in Section IV.

Definition of objectives. The literature and regulation advocate that for a user to have trust in a system they must have clarity on the data the system holds, the underlying mechanisms, data storage and access controls [22] as well system's functionalities. We highlight some of the statements that led to these qualitative objectives [37]. The GDPR takes a user-centred approach where data controllers must implement transparency in their data processes [38]. However, "the right to be informed" is often violated as users' personal and behavioural data is collected without their knowledge of why or how it is being used [11, 27]. This can cause users to lose trust or have false trust in a system [31]. Similarly, users require knowledge about the system's functionality. Therefore, it is important to address transparency at the requirements level [13, 39]. In this work we focus on different aspects, such as functions and data, of a requirements specification, and produce specific patterns (Data, GDPR, Use Case, Process) to improve these specifications and add missing requirements.

Design and development. TEM is a technical solution for creating transparency requirements supporting users trust in software using prescriptive knowledge [40] . In designing TEM we took into consideration the factors suggested to enhances users' trust judgement, i.e., information relevance and user expectation [15, 41]. We employ user goals to judge information relevance and the addressed expectations by generating transparency requirements for different system aspects. We argue that to provide the relevant information to the user, transparency must improve requirements specifications. We use patterns as a standard software engineering tool to capture solutions to recurring problems. Requirements patterns can be used during early stages of software development and support re-usability, consistent vocabulary and enhanced communication [42] while addressing the issue of incomplete requirement [32]. TEM patterns are designed following Withall requirements patterns [43]. We provide patterns for all aspects we need to be transparent about, i.e., data, operations, and processes, orthogonally distinguish static from dynamic patterns depending on the nature of the information to be shared. How to best present such information falls outside the scope of the patterns, see research on user-friendly representations of privacy-related information and requirements design [38, 44]. TEM is structured so that patterns are decoupled and can be applied across different locations and at different times by different teams of experts. TEM gives the developers options to specify whether the user should receive the information directly (push) or needs to ask for it (pull). The methodology is designed to be executed by requirements engineers and developers. The transparency patterns provoke them to think about what is missing in their requirements, select, and check user goals, conditions, and data interests (see below) where transparency is needed. This also helps to detect omissions in existing requirements.

Figure 1 outlines TEM, its conceptual framework and requirements patterns, highlighting TEM's steps from left to right. The specific steps are described in Section III.B. The engineer generates requirements using transparency patterns. These patterns cover the different types of functional requirements in the software see Section III.A.

A. *Transparency Requirements Patterns*

The methodology is driven by patterns covering three distinct aspects of the system, i.e., data, use cases, and processes, each with their own means of transparency. Each aspect is supported by a set of patterns. Data-Driven Patterns (DDP) include static (type-level) patterns which describe the information related to the data schema identified by classes, attributes, and associations, and dynamic (instance-level) patterns covering the actual data state, e.g., to inform the user

what data is currently stored about them and when it is being used.

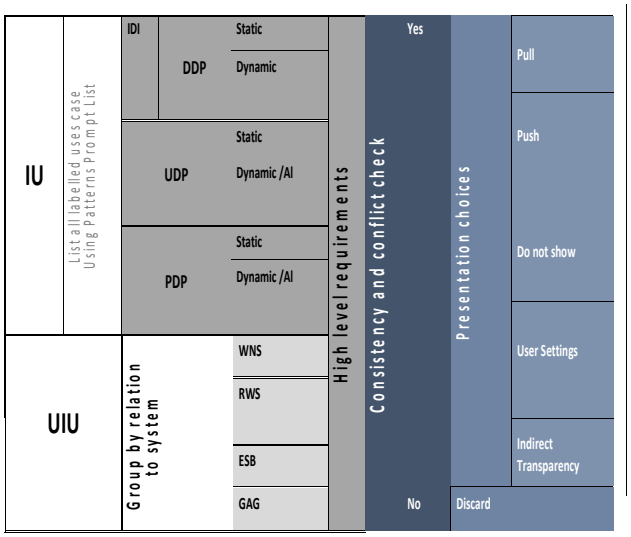

Figure 1. Overview of TEM. (IU = Implemented Use cases, UIU = Unimplemented Use cases, IODI = Identification of Data Interest, DDP = Data Driven Patterns, UDP = Use Case Driven Patterns, PDP = Process Driven Patterns, WNS = Why Not in System, RWS = Relation With System, ESB = Explain System Boundary, GAG = Give Alternative Goal) The conceptual framework and the requirements patterns are described in sections A and B.

Use Case and Process Patterns are also sub-divided into static and dynamic ones. For example, a static process pattern provides general information about a particular process while a dynamic one reports on its execution, including its control state or enabled actions. Additionally, both domains have alternative patterns that identify which enabled and available alternative sequences of processes or use cases the system needs to reveal to the user in case the main path of execution may fail. GDPR-driven transparency patterns are special data-driven ones derived from GDPR requirements. They include the Data Protection and Data Subject Rights patterns, these are explained in more details in our related work [45]. The first generates transparency requirements for data under data protection legislation the system holds about the user. The second pattern provides information about the users' rights on that data. We define the following patterns:

1. Data Transparency
    a. Static Data Transparency Pattern
    b. Dynamic Data Transparency Pattern
2. GDPR Driven Transparency
    a. Data Protection Transparency Pattern
    b. Data Subject Rights Transparency Pattern
3. Use Case and User Story Transparency
    a. Static Use Case Transparency Pattern
    b. Dynamic Use Case Transparency Pattern
    c. Alternative Use Case Transparency Pattern
4. Process Transparency
    a. Static Process Transparency pattern
    b. Dynamic Process Transparency Pattern
    c. Alternative Process Transparency Pattern

Each pattern demonstrates how to write the resulting requirement. Some patterns specify conditions when to be transparent. Conditions are logical statements composed of combinations of data operations with actors performing them, of the form «actor» «data operation» «data type».

For instance, in a condition "system admin updated address", "system admin" is the actor, "updated" the data operation and "address" the data type.

In the following we show three transparency requirements patterns with simplified outputs to explain the methodology. For a complete list of the patterns see supporting materials[1].

Static Data Transparency Pattern:

Manifestation and name	Standard Static Data Pattern
Domain	Data
Applicability	Use to provide information about the *data types* the system holds, the actions available on these data types and by which actors they can be performed.
Contents	Data interest. Data type. System actions and their actors.
Derivation of Contents	The data interest is defined by the requirements engineer based on user concerns and regulation. Data types are identified by labelling them with data interests they belong to. Checking the existing use cases that deal with this data type, a list of actions and a list of actors performing them is created.

Summary	**Definition**
Template for «data type»	If *«data type»* belongs to *«data interest»* The system must communicate to the user that it holds data of *«data type» on which «list of actions»* can be performed by *«list of actors»*

An application of the pattern results in an instantiation, e.g.:

Summary	**Instantiation**
Template for Address data	The system must communicate to the user that it holds data of type *Address* on which *update actions* can be performed by the *system administrator*.

Data Subject Rights Transparency Pattern:

[1] https://github.com/Bara60/Supplementary-info-Transparency.git

Manifestation and name	Standard Data Subject Rights Pattern
Domain	Data GDPR
Applicability	Use pattern to provide information about what data system holds about the user and their rights on that data under GDPR.
Contents	Data interest. Data types. Data subject's rights.
Derivation of Contents	The data interest is defined based on user concern or GDPR, any detail deemed relevant to data subjects including but not limited to personal data; can include an area of personal data (e.g., shipping data combines name and address), or other concerns (e.g., choices on restricting data process). Data types: the system's data schema (classes, attributes, associations); also referred to as categories of data by the regulation (Art.15(1)(b)). Data subject's rights: the rights described by the GDPR (Art. 15 to 22), such as the right to access, to rectification, to erasure, to data portability, and others.

Summary	Definition
Template for «data type»	If *«data type»* is a *«data interest»* the system must communicate to the user that it holds data of «data type» and how «*data subject's rights*» can be performed by data subject.

Summary	Instantiation
Template for Address data	The system must communicate to the user that it holds data of type *Address* and how actions of *erasure, restriction of processing,* and *data portability* can be performed by the user (data subject).

Alternative Use Case Transparency Pattern:

Manifestation and name	Standard Alternative Use Case Pattern
Domain	Use case
Applicability	Use pattern to identify which enabled and available use cases the system needs to reveal to the user in case of a failure to meet pre/ postconditions.
Contents	Pre and postconditions of the use case. Use case. List of Alternative use cases. Reason why the system has chosen this sequence.
Derivation of Contents	Pre/postconditions from use case descriptions. Use cases with alternative sequences. Alternative (available and enabled) use cases in the current state and/or ways to fulfil the pre/postconditions of the current use case.

Summary	Definition
Transparency about «use case» because of unmet «pre/post conditions»	If current system state does not meet «pre-conditions "before" | post conditions "after» of «use case» the system must communicate « list of alternatives» for «use case» because of «reason».

Summary	Instantiation
Transparency about "transfer money to account" because of unmet condition "enough funds in account".	If current system state does not meet the "enough funds in account" condition at start of "transfer money", the system must communicate "deposit money into bank account" as alternative action. Because of fund below 40 pounds.

B. *Applying the Methodology*

The methodology consistent of four steps preceded by the preliminary step of identifying data interests. Based on these, the first step applies the transparency patterns. The second step focuses on the goals that are out of scope for the current implementation of the system and the foreseeable future. The rationale behind this step is to manage the users' expectations, making them aware of limitations in relation to their goals. The third step is a consistency check for the resulting requirements.

Step 0: Identify Data Interests.
Input: User data concerns based on goals, laws and regulations.
Output: List of data interests.

To ensure consistency of requirements, it is important to identify data interests prior to applying the patterns, so that the same data interest used in different areas of the application can be grouped and used in the same way. Data interests are a reference to data that are defined under the same concern or satisfying the same use goal, e.g., persona data, performance data, account data, etc. This process provides the flexibility to identify any data group related to the user. The process of generating data interests is as follows:

1. The engineer identifies data in the system by a name, e.g., "personal data" as defined under GDPR [11].
2. They select the data types that are part of the data interests by choosing the relevant data classes of the conceptual data model.

For example, users' concern about data is often shown in their goals e.g. "I want to see my shipping address before

confirming payments". An engineer in this case creates a data interest called "shipping data" to represent this user data concern.

Step1: Apply transparency patterns to implemented goals.
Input: List of user goals that are (to be) implemented.
Output: Complete transparency requirements document (TRD).

We map the goals to one or more of the pattern categories (Data, GDPR, Use Case, Process). Goals can refer to functionality already in the system, or to be added in the next iteration. Based on the use cases or user stories selected we use the pattern prompt list to choose the corresponding patterns. For each data interest from step 0, identify the corresponding data patterns and generate the requirements following the pattern templates. Add the resulting pattern instantiations to the transparency requirement documentation.

For example, from a user story such as "As a user, I want to make an online purchase using quick checkout and have it shipped to my current address, so that I can order as fast and hassle free as possible", our patterns will generate the following requirements:

GDPR Data Protection pattern: The system must communicate to the user that it holds data of type Address which is "accessed by delivery service" and "This data is encrypted and will be stored until your account is deleted".
Static Use case pattern: The system must communicate to the user that "default address must be given before quick checkout".

Step 2: Add justifications for non-implemented goals.
Input: List of all non-implemented goals (e.g., from development backlog)
Output: List of transparency requirements relating to non-implemented goals.

We assign each goal one of the following justifications: WNS = Why Not in System, RWS = Relation with System, ESB = Explain System Boundary, GAG = Give Alternative Goal and record a reason why that justification has been assigned. The goals are then clustered by reason to create a single transparency requirement to inform the user why a particular goal has not been implemented. This gives the user a better understanding of the limitations of the system and improves their mental model. As these goals are not in the focus of the system, transparency about them is usually achieved through "pull transparency" where the user must actively call for the information.

For example, in food delivery systems a user goal can be "I want to order my groceries". The justification here would be ESB where the system currently only delivers fast food because both fast food and groceries are food types.

Step 3: Consistency check
Input: TRD (from Step 2*)*.
Output: Finalized TRD.

We finalise the transparency requirements document by applying a consistency check that results in removing duplicates and conflicting requirements, as well as aggregating requirements that can be combined. The requirements engineer needs to check if there is any information disclosure or conflict with the other requirements, relating to security or privacy.

For example, sharing information about data servers could cause a security breach.

IV. Demonstration and Evaluation

To demonstrate the use of TEM, a case study was conducted on a software project by Spirit Healthcare, with a qualitative analysis of participants' questionnaires and a comparative review of the output of TEM. Spirit Healthcare is a health organisation with a small technology branch. They handle sensitive and individually identifiable medical data across several applications, including remote patient monitoring, education booking, and management. The participants were team members, and include domain experts, senior and junior developers. Participation in the case study and evaluation was voluntary. The evaluation used semi-structured interviews, focus groups, and pre- and post-questionnaires.

The team consists of four developers (D1 to D4) one of which is the project manager, scrum master, and lead architect (DE4). The team also has a domain expert (DE5) with a joint role of product manager who sits between the development team and the business. As it is common in agile development [46], in our evaluation developers and domain experts act as spokespersons of the end user. This is necessary because trust is a subjective and individual concept based on a range of different factors [1, 47]. Such confounding factors would need to be measured and accounted for to isolate TEM's impact on trust. Also, to be effective the resulting requirements must be refined considering usability (i.e., how the information is best presented to the user) which is beyond the focus of our research but will impact on the effectiveness of transparency and thus on user trust. The use of domain experts as proxies allows to circumvent this problem in a qualitative study.

Prior to the main case study, a pilot study was conducted with experts from requirements engineering, privacy, and data protection. This resulted in a refinement of the presentational and implementation structure of TEM and the GDPR patterns addressing recent legislation. TEM has been developed to aid domain experts and requirements engineers, but it became clear that it also helps developers in defining transparency requirements and detailing existing requirements. As a result of the preliminary study, the choice of this company was confirmed with developers and domain experts selected as stakeholders responsible for requirements engineering.

The approach of the case study and its evaluation was to identify instances of good implementation of TEM by experienced developers. The evaluation was divided into three stages: formalisation and exploration, training and trial, and execution. These included introducing the team to the process,

gathering information on the team structure and particular problem area, training the team members, and implementing the methodology.

Design science research targets a knowledge gap by posing research questions [48, 49]. Hense, to evaluate TEM we assess our objectives based on the results of using TEM [37]. Then we use a "how" question for the design process of TEM and a "what is" question for knowledge about its impact [49]. We thus formulate the questions:

- *How does TEM help in defining general requirements and, specifically, transparency requirements?*
- *What is the impact of TEM on trust judgment and other system qualities?*

These questions give an indication of both the execution of TEM and its usefulness.

The evaluation used several data collection methods. Pre-questionnaires gathered data on the team's structure and the context in which the methodology was to be implemented. Semi-structured interviews consisted of an open conversation with prompt questions to discuss how the methodology was run, where it worked effectively, and where it was less effective. This was done in two phases – in a group and with individuals – to allow more open conversations and develop considerations and points of interest. The discussions were recorded and associated with themes. Later, two types of post-questionnaires were used, the first including technical questions and the second eliciting opinions on the impact of TEM on trust and other system qualities. Finally, the output from TEM was compared with the original user story definitions to assess the quality of the resulting requirements, in relation to transparency.

Points of interest to structure the analysis of responses were derived from the questions above and the literature. To analyse the data, thematic coding was used to group the responses. [50]. These themes include Transparency of Available Software Functions (TF), Transparency Data-Related Issues (TD), Quality of Documentation (QD), Missing Requirements Being Covered (MR), Limitations, two additional themes, Defined Requirements (DR) and Informing the User (IU). They assess how transparency can be introduced or must be present in the system requirements. Effectiveness was measured by evaluating the extent to which these themes are covered, and how TEM users have adopted the methodology.

A. Results

Across the user stories of the chosen application TEM has generated the 64 new transparency features and 28 refinements to existing functional and data concern. The domain experts completed their post-questionnaire based on their knowledge about the system's users. The questions focused on the expected impact on the users' trust of implementing the resulting requirements the following trust factors and system qualities:

1. *Users' cognitive trust: understandability, reliability, and technical competence.*
2. *Usability and usefulness.*
3. *Temporary and permanent reassurance, data traceability*

The responses were all positive, emphasizing that transparency "will lead to traceability throughout the application". The participants reflected in their answers, how TEM supports achieving transparency and why this helps users trust in the system. The domain expert noticed that TEM influences the way they think about the experience of the user "TEM prompts me to think about the user experience from a perspective of trust". Trust was equated with making "a user feel very confident about the system" (quote from DE4, but also voiced by DE5). To achieve user confidence, the system needs to give "feedback […] after carrying out commands" (DE4) and inform the users of their actions (DE5). But you need to be careful, because while "some users would feel reassured" (DE4) if all the TEM outcomes are implemented, others might "feel like the system is trying to tell them too much" (DE4). This concern is addressed in TEM by offering the specialist applying the method to choose from different options on how to achieve transparency (push, pull, do not show, user settings). Especially the option for "user settings" leaves the users the freedom to decide, which level of information is right for them. However, selecting this transparency option needs to be balanced with the number of settings that are presented to the user as well as development time and effort restrictions.

4. *Users making informed decisions, building their trust, and meeting their expectations.*

Domain experts also stated that when applying TEM users' trust can be built by enabling them to make informed decisions. DE5 noted that TEM will help the user to decide if they want to use the software or not. DE4 mentioned that in our domain area the users tend to assume things and build false trust based on them. Transparency in this case is an important requirement. DE4 added that taking into consideration that users differ, TEM will help in meeting user expectation when the system responses "validate what the user did whenever they use any part of the system".

5. *Methodology characteristics.*

The domain experts shared their insights into TEM's characteristics: "The methodology is concise, consistent and replicable." DE5. Another domain expert noted that TEM works like a "safety net" for developers writing their software specifications following user stories, reminding the team of what system aspects to consider. TEM also "serves to generate discussion between developers/stakeholders around points that may not otherwise be considered" (DE4).

Transparency of Available Software Functions (TF): D4 pointed out that "Regarding other systems that I have worked with, I think there are clear benefits to a process that ensures consideration is given to explaining potentially complex tasks that the system is capable of to the user. This kind of extra detail is often not picked up by developers, so being picked up by the methodology would be helpful". They added that

User story. Display Structured Education Courses

As a patient I need to be shown the appropriate SE courses for me. This will include courses available online (referred to as Remote) and face to face group courses. I will need to see the detail of what the course is offering, where the course is being held and the date and time…

Static Data requirement result:

The system must communicate to the user that it holds data of type postcode and locational data which will be sent to Google via google maps and stored on their servers.

Dynamic Use case requirement result:

In case the patients search parameters are do not return results the system must communicate to the user that the search parameters are too narrow and that if they widen them to a particular degree more results would become available.

GDPR requirement result:

The system must communicate that Uses Postcode to search for course and is only "accessed by system admin". This data is anonymized and will be stored until your account deletion.

Figure 2. Example user story – (Static data, Dynamic Use case and GDPR).

complex features are "better explained, then it attracts more usage. This is an example of a user trusting the system to do what it says it should. Once that detail was added, users had confidence in customising that part of the system". D3 explained that TEM helps in maintaining user-focused development: "At the task implementation level it is all but lost in a more functional description." So much of current software development is centred around user focused development especially the elicitation process but this link can be lost when actually under development. D3 is alluding to centring the requirements around communicating with the user as the user stories are broken down into tasks. As a result, TEM can bring user perspective right to the point of writing code. Figure 2 illustrates an original user story with resulting requirements after applying TEM.

Transparency Data-related Issues (TD): The experts stated how important it is to report to the user what data is being collected about them. D3 reported using TEM helps them to consider the data points and interests of the system tasks and followed that up by referring to an issue where the system collects data without telling the user "Made me consider each of the data points and data interests for each task and structure the task description around them. We often collect data without telling the users what it is being used for, so it was good to considered why at each point in the application we are collecting given things. Makes you rethink the validity in collecting certain things, such as user personality ethnic details were removed from the system as we had no real reason to collect it". This indicates how necessary it is to justify why the user's data is collected and in case there is no justification, clearly the data should not be collected. The developers stated that the data interests help in grouping all the data related information, then added "Data interests are a great extension on the concept of data tables – you can group many tables under a data interest".

Quality of Documentation (QD): Regarding the quality of documentations of TEM (easy to use/reuse and maintain), developers noted that it is hard to follow at the start and became easier once more familiar. D3 stated that "Good as a whole, can be difficult to follow at times" however, "found the process got a lot easier as time went on and the process naturally streamlined itself as I become more familiar with the process and documentation". Also, D4 noted that "I felt that it could do with more complete examples".

Missing Requirements Being Covered (MR): The developers have stated in their post-questionnaires that the methodology helped them to cover the missing requirements in their example applications. Developer D3 stated that the methodology "helps fill in many of the gaps particularly around informing the user". Developer D3 added that the user is usually only informed in errors cases, and that creates "lots of gaps as it was often down to the developer's discretion" they mentioned that now with TEM" the structure forces user to be informed about everything so you don't miss the important communications".

Defined Requirements (DR): D1 stated that "We don't always have the system aspects required for the task defined and available". D3 was very affirmative on that TEM "Helps to define the functionality more clearly. And has a user focused bent with everything focusing on communication with the user which is often an afterthought for me." Developer DE4, stated that the methodology helps in defining the requirements "since there is more thought given to the development items."

Informing the User (IU): The developers have stated that TEM helps to sustain their user focus. One of the developers noted that TEM "helps to maintain user-focused development later into the development cycle where at the task implementation level it is all but lost in a more functional description." They elaborated further "This meant we informed the user about processes we probably wouldn't have bothered before with but can be informative. On data collection every input is more deeply scrutinized especially peripheral data collection processes which were often overlooked". D4 also stated that applying the methodology aids in informing the user about their data and other system aspects as it "raises the thought and discussion between stakeholders, which then if carried out will certainly help the user in their use of the system." They highlighted that TEM aids them to reflect on the data that is stored and, how to communicate to user. During the focus group one of the team members said that TEM brings the user to the implementation and the user stories "by letting us think of what the user wants to know from each systems aspect been addressed in the user story e.g., data, operation". One of the developers describes "user centric view" with regards to transparency concerns mostly relates to the user information that they impart. However, a big part of personally identifiable collected data is gathered passively via behavioural, statistical and usage data. This is often not identified as a data concern and the user is rarely directly informed.

Limitations: The developers and the domain experts noted several limitations of TEM. They have been asked directly in the post-questionnaires how you think the methodology can be

improved. The domain expert found that the time to execute TEM can restrict its use as DE4 stated "The limitations from my perspective really were time-based" and based on the "the circumstances of the development team as to whether they could fit this detailed analysis into their plans" (DE4). Their suggestion was to bring more complete and concise examples so that the developers have better expectations and comprehend what it is needed to be used for. Additionally, D3 noted that the transparency information can overwhelm the end user, depending on the type of the user "and the amount of information that they want to see". The solution to that is by specifying different types of users, however, this means that this option needs to be clearer in the condition section. DE4 also noted on step two of the methodology where the unimplemented system related goals are clustered, by the reasons why they are not implemented for the user to be informed about them, it is not necessary as no data of these goals would be recorded or carried out after they have discarded. However, it is important to note that the developers have not run the presentation choices themselves in this evaluation which potentially help in how to communicate the information to the user over the application. This issue is partly addressed by the concept of conditions where the communication of certain information to the user needs to be related to their current interaction with the system. However, the team did not use the concept to its full potential. At some point in the training, it was important to remind developers to use conditions. Moreover, the current version of TEM allows the engineer to have the final decision on how the information is communicated to the user with the help of presentation choices.

B. Threats to Validity

We discuss threats to the validity of the demonstration and evaluation of TEM as presented in this section which could render the evaluation results invalid and explain the extent to which we were able to mitigate them. Weaknesses of TEM itself are discussed in the following section V.

Although the evaluation was conducted with an entire development team, this team only consisted of 5 members. This could affect the range of answers, where responses are subjective relying on the members past and current experience. The project chosen during the last stage of the evaluation was selected because it was concerned with system aspects where transparency is vital as well as difficult, including sensitive medical data. The application was small, consisting of 29 user stories. This was a manageable size but might have not stretched the methodology to its full capacity to discover all its limitation. As a result, the scalability of the methodology was not tested.

In this case study domain experts' opinions indicated how TEM enhances user trust. A more complete evaluation of the methodology should include measuring the impact on trust directly, but this requires a longer-term study with many participants to be able to control confounding factors and an extension of TEM towards user interaction design to define how the information is best presented to the users, see recent work by Murmann et al. [44].

The methodology has only been evaluated within the company's current development practices. This makes it difficult to distinguish between the impact of TEM and, e.g., of the methods and tools used by the company. The interviews and focus groups helped distinguish the two. Further, TEM was only compared to the baseline agile approach, which we deemed acceptable as an industry standard.

This evaluation used qualitive analysis methods; however, they have several limitations. For instance, results can be subjective, involving few participants, and thus have limited generalizability [51]. Another weakness when evaluating the results with a specific company is that the answers can be biased and effected by the company environment. For instance, some responses from the management team can be diplomatic. This may not happen in a lab environment. To overcome these weaknesses, the methodology could be trialled on a larger scale across multiple companies and projects.

More fundamentally, our research questions "how" and "what is" define our investigative direction, therefore delimiting the perimeter of analysis [52]. However, by phrasing the questions in an open-ended and general way, rather than specific hypotheses, we follow best practices of qualitative research.

V. DISCUSSION

This study shows that TEM provides a capable framework for engineering transparency requirements. Once the transparency requirements have been defined, engineers and developers can decide what information to show or hide by carrying out a consistency check against other requirements.

To provide the relevant information to the user, transparency must be incorporated into functional requirement specifications. The methodology uses the projects' high-level user stories and, depending on data interests and processes, generates requirements ranging from informing the user on who has access to their data to what processes are occurring. TEM thus guides developers and engineers to questions of transparency and trust and allows them to build transparency and trust into the system from the start. If applied in a small mixed team like ours, TEM is useful for bringing the users' view into the process from requirements to implementation. The resulting requirements on data, use cases, and processes are described in terms of communication with the user. This focus gives the requirements a holistic nature.

Data interests are a way of grouping similar user data concerns. The complexity of many applications means that data is acted on in multiple locations within an application. All the developers recognised the benefit of creating data interests as a useful way of grouping data types. For example, a data interest such as "user data" could include any data that can identify the user, e.g., personal data, behavioural data, and so grouped into a data interest helps maintain a consistent approach to handling transparency on data. The creation of data interests should be

an iterative process distributed throughout the team via a top-down approach to maintain consistency.

Our pattern-based approach addresses gaps in transparency from early requirements onwards. TEM can directly help to fill these gaps by identifying discrepancies between pre- and post-conditions, data, and processes, creating more complete and consistent specifications.

TEM generates transparency about alternative (exceptional, error-handling, or recovery) processes and their use of data, which is often ignored by general requirements engineering practices. The evaluation revealed that, how data was used in alternative processes was all but completely ignored in the project's initial elicitation process. This indicates that TEM can support the quality of requirements more generally.

TEM is designed to fit agile practices and management tools. The case study with Spirit Healthcare demonstrates that the pattern-based approach is suitable in such contexts. The team uses Azure DevOps to manage their development life cycle, organising user stories into product backlog items, which are then broken down into development tasks. The implementation of the methodology was reported to fit well within this process. Research in requirements engineering shows that, because of the focus on rapid iteration and results, requirements in agile development are often not well defined, and that the users' perceptions are not fully considered [53]. TEM addresses this issue by extending the agile process so teams can continue to work inside their familiar structure while more consciously addressing the users' concerns.

Our study also shows that TEM improves data minimization and exposes information that is not directly accessible to the user, thus ignored by standard requirements elicitation methods. TEM supports both through systematically reflecting on the purpose for data collection and the use of that data.

The study also revealed weaknesses of TEM. The results from the training and the trial stages show that participants in their first attempts seemed to complete the requirements from their knowledge of that system aspect, without necessarily following the pattern. As more of the user stories were elicited, there was a substantial improvement of the quality of the output and adherence to the framework. This shows that there is a learning curve for implementing the methodology. The effect was that, at first, developers tended to create their own templates to capture their intended requirements because they did not understand the structure and found it hard to fit it to a particular use case. As they spent more time with the methodology, they described the process as faster and clearer.

There was a concern regarding inconsistent outputs from different patterns and data interests. To rationalize the use of patterns as a way of assisting developers in handling data interests, these should be defined first, confirmed by the lead developers and domain experts, before being used by the team. This hierarchical process had to be made clearer in the methodology description.

More knowledge about the problem area might be required to get the most out of the engineering methodology. There are consistency checks by developers and engineers where they must decide what information to display to the user, what information to be pushed or pulled, as well as checking when information disclosure might cause data breaches. It is important that the participants know their targeted audience and application domain to make these decisions.

The patterns are designed to generate requirements separately from each system aspect, but this could cause redundancy where such aspects overlap, and thus inconsistency. To rationalize the use of patterns as a way of assisting developers in handling data interests, these should be defined first, confirmed by the lead developers and domain experts, before being used by the team. This hierarchical process had to be made clearer in the methodology description.

VI. Conclusion and Future Work

TEM has been shown to systematically generate transparency requirements for informing the user about how data is being used and who has access to it. This degree of transparency is extended to backend and passive data collection processes where informing the user is often overlooked. Although the methodology takes effort to instantiate and learn, it produces holistic user stories as illustrated by the quality of the output and comments from the developers and domain. The responses and the technical results show that the methodology is useful for defining requirements more completely and consistently.

The task of systematically integrating the resulting transparency requirements into the implemented software system is yet to be trialled and evaluated. TEM is an initial step in this integration.

The structure of TEM helps in generating data interests top-down. Hosseini el at. [34] mentions a "hierarchy of importance and effectiveness of information quality dimensions on the perception of transparency, and their impact on the decision making processes" [54] which could be used to prioritise data interests and thus guiding the generation of transparency requirements.

There are several aspects to this research to be investigated further. Currently, transparency of software is solely down to company discretion and thus mainly driven by commercial considerations. Future work could aim to standardize transparency principles for privacy and safety goals where companies or systems will only be certified as transparent if they adhere to the defined practices. Some recent studies suggest a movement in this direction [55]. Future work could include TEM being standardized for project management tools such as Microsoft DevOps, which could improve consistency issues stated as limitations. Project management tools represent an amalgamation of individuals, processes, and products to ensure uninterrupted distribution of value to users [56]. They are therefore good platforms on which to integrate transparency engineering.

REFERENCES

[1] B. Zieni, R. Chitchyan, and R. Heckel, "Trust as a sustainability requirement," in *Proceedings of the 6th International Workshop on Requirements Engineering for Sustainable Systems (RE4SuSy 2017), Lisbon, Portugal*, 2017, vol. 5.

[2] J.-N. Lee, M. Q. Huynh, and R. Hirschheim, "An integrative model of trust on IT outsourcing: Examining a bilateral perspective," *Information Systems Frontiers,* vol. 10, no. 2, pp. 145-163, 2008.

[3] M. Söllner, A. Hoffmann, H. Hoffmann, and J. M. Leimeister, "How to use behavioral research insights on trust for HCI system design," in *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, 2012, pp. 1703-1708.

[4] A. Hoffmann, M. Söllner, H. Hoffmann, and J. M. Leimeister, "Towards trust-based software requirement patterns," in *2012 Second IEEE International Workshop on Requirements Patterns (RePa)*, 2012: IEEE, pp. 7-11.

[5] F. Moyano, C. Fernandez-Gago, and J. Lopez, "Building trust and reputation in: A development framework for trust models implementation," in *International Workshop on Security and Trust Management*, 2012: Springer, pp. 113-128.

[6] P. Madhavan and D. A. Wiegmann, "Similarities and differences between human–human and human–automation trust: an integrative review," *Theoretical Issues in Ergonomics Science,* vol. 8, no. 4, pp. 277-301, 2007.

[7] M. Zloteanu, N. Harvey, D. Tuckett, and G. Livan, "Digital Identity: The effect of trust and reputation information on user judgement in the Sharing Economy," *PloS one,* vol. 13, no. 12, p. e0209071, 2018.

[8] J. Huang and M. S. Fox, "Trust judgment in knowledge provenance," in *16th International Workshop on Database and Expert Systems Applications (DEXA'05)*, 2005: IEEE, pp. 524-528.

[9] L. M. Cysneiros, "Using i* to Elicit and Model Transparency in the Presence of Other Non-Functional Requirements: A Position Paper," in *iStar*, 2013: Citeseer, pp. 19-24.

[10] Y.-C. Tu, "Transparency in software engineering," ResearchSpace@ Auckland, 2014.

[11] D. Spagnuelo, A. Ferreira, and G. Lenzini, "Accomplishing Transparency within the General Data Protection Regulation," in *ICISSP*, 2019, pp. 114-125.

[12] H. Felzmann, E. F. Villaronga, C. Lutz, and A. Tamò-Larrieux, "Transparency you can trust: Transparency requirements for artificial intelligence between legal norms and contextual concerns," *Big Data & Society,* vol. 6, no. 1, p. 2053951719860542, 2019.

[13] J. C. S. do Prado Leite and C. Cappelli, "Exploring i* Characteristics that Support Software Transparency," in *iStar*, 2008, pp. 51-54.

[14] M. Hosseini, A. Shahri, K. Phalp, and R. Ali, "Foundations for transparency requirements engineering," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2016: Springer, pp. 225-231.

[15] R. F. Kizilcec, "How much information? Effects of transparency on trust in an algorithmic interface," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 2390-2395.

[16] U.-V. Albrecht, O. Pramann, and U. von Jan, "Synopsis for health apps: transparency for trust and decision making," in *Social media and mobile technologies for healthcare*: IGI Global, 2014, pp. 94-108.

[17] S. Moghaddam, M. Jamali, M. Ester, and J. Habibi, "FeedbackTrust: using feedback effects in trust-based recommendation systems," in *Proceedings of the third ACM conference on Recommender systems*, 2009, pp. 269-272.

[18] D. J. Kim, D. L. Ferrin, and H. R. Rao, "A trust-based consumer decision-making model in electronic commerce: The role of trust, perceived risk, and their antecedents," *Decision support systems,* vol. 44, no. 2, pp. 544-564, 2008.

[19] U.-V. Albrecht, "Transparency of health-apps for trust and decision making," *Journal of medical Internet research,* vol. 15, no. 12, p. e277, 2013.

[20] H. Cramer *et al.*, "The effects of transparency on trust in and acceptance of a content-based art recommender," *User Modeling and User-adapted interaction,* vol. 18, no. 5, p. 455, 2008.

[21] P. Murmann and S. Fischer-Hübner, "Usable transparency enhancing tools: A literature review," 2017.

[22] P. Murmann and S. Fischer-Hübner, "Tools for achieving usable ex post transparency: a survey," *IEEE Access,* vol. 5, pp. 22965-22991, 2017.

[23] M. Turilli and L. Floridi, "The ethics of information transparency," *Ethics and Information Technology,* vol. 11, no. 2, pp. 105-112, 2009.

[24] N. Bevan and M. Azuma, "Quality in use: Incorporating human factors into the software engineering lifecycle," in *Proceedings of IEEE International Symposium on Software Engineering Standards*, 1997: IEEE, pp. 169-179.

[25] L. M. Cysneiros and V. M. B. Werneck, "An Initial Analysis on How Software Transparency and Trust Influence each other," in *WER*, 2009.

[26] M. Hosseini, A. Shahri, K. Phalp, and R. Ali, "A modelling language for transparency requirements in business information systems," in *International Conference on Advanced Information Systems Engineering*, 2016: Springer, pp. 239-254.

[27] R. Meis, R. Wirtz, and M. Heisel, "A taxonomy of requirements for the privacy goal transparency," in *International Conference on Trust and Privacy in Digital Business*, 2015: Springer, pp. 195-209.

[28] R. Bachmann and A. Zaheer, *Handbook of advances in trust research*. Edward Elgar Publishing, 2013.

[29] "Stakeholders not satisfied with probability assessments." (accessed 09/10/2018.

[30] Y.-C. Tu, C. Thomborson, and E. Tempero, "Illusions and perceptions of transparency in software engineering," in *2011 18th Asia-Pacific Software Engineering Conference*, 2011: IEEE, pp. 365-372.

[31] H.-H. Herrnfeld, "Article 67 Data protection by design and by default," in *European Public Prosecutor's Office*, 2020: Nomos Verlagsgesellschaft mbH & Co. KG, pp. 513-514.

[32] C. Palomares Bonache, "Definition and use of software requirement patterns in requirements engineering activities," in *Proceedings of the REFSQ 2011 Workshops, the REFSQ 2011 Empirical Track, and the REFSQ 2014 Doctoral Symposium*, 2014, pp. 60-66.

[33] M. Hosseini, A. Shahri, K. Phalp, and R. Ali, "Engineering transparency requirements: A modelling and analysis framework," *Information Systems,* vol. 74, pp. 3-22, 2018.

[34] S. M. Hosseini Moghaddam, "Engineering of transparency requirements in business information systems," Bournemouth University, 2016.

[35] K. Schwab, A. Marcus, J. Oyola, W. Hoffman, and M. Luzi, "Personal data: The emergence of a new asset class," in *An Initiative of the World Economic Forum*, 2011.

[36] H. A. Simon, "The sciences of the artificial 3rd ed," ed: MIT Press Cambridge, 1996.

[37] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems,* vol. 24, no. 3, pp. 45-77, 2007.

[38] A. Rossi and G. Lenzini, "Transparency by design in data-informed research: A collection of information design patterns," *Computer Law & Security Review,* vol. 37, p. 105402, 2020.

[39] C. Cappelli and J. Leite, "Software transparency," *Business and Information Systems Engineering,* vol. 2, pp. 127-139, 2010.

[40] S. Gregor and A. R. Hevner, "Positioning and presenting design science research for maximum impact," *MIS quarterly,* pp. 337-355, 2013.

[41] S.-Y. Chien, M. Lewis, K. Sycara, A. Kumru, and J.-S. Liu, "Influence of culture, transparency, trust, and degree of automation on automation use," *IEEE Transactions on Human-Machine Systems,* vol. 50, no. 3, pp. 205-214, 2019.

[42] F. Loizides, M. Winckler, U. Chatterjee, J. Abdelnour-Nocera, and A. Parmaxi, "Human Computer Interaction and Emerging

Technologies: Workshop Proceedings from the INTERACT 2019 Workshops," ed: Cardiff University Press, 2020.
[43] S. Withall, *Software requirement patterns*. Pearson Education, 2007.
[44] P. Murmann and F. Karegar, "From Design Requirements to Effective Privacy Notifications: Empowering Users of Online Services to Make Informed Decisions," *International Journal of Human–Computer Interaction,* pp. 1-26, 2021.
[45] B. Zieni, D. Spagnuelo, and R. Heckel, "Transparency by default: GDPR Patterns for Agile Development," in *The 10th International Conference on Electronic Government and the Information Systems Perspective (EGOVIS2021),Linz, Austria*, 2021 (forthcoming).
[46] A. de Ste Croix and A. Easton, "The product owner team," in *Agile 2008 Conference*, 2008: IEEE, pp. 274-279.
[47] X. Luo, H. Li, J. Zhang, and J. P. Shim, "Examining multi-dimensional trust and multi-faceted risk in initial acceptance of emerging technologies: An empirical study of mobile banking services," *Decision support systems,* vol. 49, no. 2, pp. 222-234, 2010.
[48] A. Hevner and S. Chatterjee, "Design science research in information systems," in *Design research in information systems*: Springer, 2010, pp. 9-22.
[49] N. H. Thuan, A. Drechsler, and P. Antunes, "Construction of design science research questions," *Communications of the Association for Information Systems,* vol. 44, no. 1, p. 20, 2019.
[50] G. Hickey and C. Kipping, "A multi-stage approach to the coding of data from open-ended questions," *Nurse researcher,* vol. 4, no. 1, pp. 81-91, 1996.
[51] J. W. Creswell, *A concise introduction to mixed methods research*. SAGE publications, 2014.
[52] A. Bryman, "The research question in social research: what is its role?," *International journal of social research methodology,* vol. 10, no. 1, pp. 5-20, 2007.
[53] M. Drury, K. Conboy, and K. Power, "Obstacles to decision making in Agile software development teams," *Journal of Systems and Software,* vol. 85, no. 6, pp. 1239-1254, 2012.
[54] A. Hemon, B. Lyonnet, F. Rowe, and B. Fitzgerald, "From agile to DevOps: Smart skills and collaborations," *Information Systems Frontiers,* vol. 22, no. 4, pp. 927-945, 2020.
[55] E. Grünewald and F. Pallas, "TILT: A GDPR-Aligned Transparency Information Language and Toolkit for Practical Privacy Engineering," in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 636-646.
[56] W. De Kort, *DevOps on the Microsoft Stack*. Springer, 2016.