Transparency by default: GDPR Patterns for Agile Development *

Baraa Zieni[1], Dayana Spagnuelo[2][0000−0001−6882−6480], and Reiko Heckel[1]

[1] University of Leicester, University Rd, Leicester, UK
{bz60,rh122}@leicester.ac.uk
[2] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
d.spagnuelo@vu.nl

Abstract. Users have the right to know how their software works, what data it collects about them and how this data is used. This is a legal requirement under General Data Protection Regulation (GDPR) and fosters users' trust in the system. Transparency, when used correctly, is a tool to achieve this. The adoption of agile approaches, focused on coding and rapidly evolving functionality in situations where requirements are unclear or fast changing, poses new problems for the systematic elicitation and implementation of transparency requirements which are driven by, but lag behind, the functionality. We propose requirements patterns addressing GDPR's principle of transparency *by default*, *i.e.*, through a systematic and structured approach based on the artefacts of agile development. We present a case study using a SCRUM process to demonstrate the effectiveness and usability of the patterns.

Keywords: transparency · GDPR · agile development · requirements patterns · trust.

1 Introduction

In 2018, new data protection rules were put in place that changed the notion of transparency in supporting user-centred approaches [25]. Transparency is one of the main principles in the GDPR[1], not only in terms of the contents of the information provided to data subjects, but also its quality and understandability. Data controllers must implement transparency in their data processes, and by doing so they help in enhancing people's trust [20].

Today, many software projects rely on agile methods like SCRUM to rapidly deliver high quality software. Unlike more traditional methods, agile approaches

* The final authenticated version is available online at https://doi.org/10.1007/978-3-030-86611-2_7

[1] *EU General Data Protection Regulation (GDPR)*: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing *Directive 95/46/EC* (General Data Protection Regulation), OJ 2016 L 119/1.

embrace change through frequent iterations of development and feedback, focusing on production code and using tests as specifications [29]. Detailed requirements engineering is mostly avoided as it entails substantial documentation [4]. A criticism of this approach is that requirements are inadequately addressed [29], recommending systematic practices such as identifying non-functional requirements [3]. If requirements are left implicit, this drives developers to implement them prematurely by "filling in the gaps". Hence, agile development with its focus on speed can cause a lack of consideration for the user [2]. We argue instead for a systematic approach to transparency requirements.

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 [5]. While trust can be addressed systematically [22], with methods including software patterns [6] and factor analysis [14], methods for engineering transparency requirements in software have not been studied in this context. In fact, transparency requirements are often underestimated during software development. This can be due to stakeholders' lack of understanding of the trust-transparency relation and worrying that users with a detailed understanding of how their data is processed may be less willing to provide it or consent to its use [23], leaving users to trust blindly in how systems handle their data [19].

We address this problem by proposing transparency patterns for requirements engineering in agile processes, designed to support analysts in applying best practices. We evaluate our proposal by a case study in a healthcare company to examine the patterns' capabilities and limitations. The evaluation answers the questions: *How has GDPR been addressed by the daily practices of agile development? How do patterns work within an agile process?* and *How do they ensure users are informed about their data concerns?* The remaining of this paper is structured as follows. Section 2 presents related work on agile development and transparency. We describe our Transparency patterns and the methodology used to create them in Section 3. Section 4 covers the evaluation and Section 5 discusses the results, concludes the paper and presents the future work.

2 Background and Related Works

In data protection law, business and governance, transparency is a user-centred principle as it refers to openness and disclosure of information to the user [25]. Transparency is seen as a meta-requirement as well as a quality in use [8]. It works at a meta level compared to functional requirements, enabling the user to know "how requirements can be fulfilled" [8]. A transparent system must provide information contextually relevant to the users' concerns actions and personal data [21]. When it is, it has been shown to elicit a high level of trust, and enhance the stakeholder-system relationship [21,7,10]. Based on this literature, transparency can be described as the appropriate amount of information that users require for better decision making and to enhance their trust.

From another perspective, in software engineering the concept of transparency can be interpreted as making the development processes of the software visible

to the stakeholders, for example, through frequent cycles of development and feedback in agile processes [26]. Such process transparency is distinct from the transparency of the end product. In this paper, we refer to the latter, transparency of the software system in the context of GDPR and agile development.

Little research has been carried out for better understanding the GDPR's practical implications in requirements engineering and software architectures. Some authors have examined transparency and its dependencies with other system goals. For instance, discussing transparency as a non-functional requirement in the context of software engineering as well as organisations: transparency knowledge is presented a graph consisting of 33 soft goals using Softgoal Interdependence SIGs [1]. Transparency has been defined as the possibility of information, intentions, behaviours to be accessed through a process of disclosure [27]. Likewise, it is considered an important concept that can support users in the decision making process. Kim *et al.* [9] states that consumers' trust, security, privacy, and perceived risk have a high influence on their purchasing decisions with websites. Trust therefore is now becoming a more crucial issue especially when it comes to stakeholders making decisions with the software.

In this research, we adopt the view that transparency leads to more informed decisions and to user trust. Therefore, we argue it is important to empower the end user, by giving them control over their data. This requires both knowledge of their rights and being informed of how their data is used in the system. This information, to be more relevant, understandable and actionable, needs to be displayed in relation to the user goals and intentions with the system. We advocate that this transparency can be engineered systematically during the development process.

While most of the papers focus on privacy requirements of GDPR (*i.e.*, [13]). The closest work presenting such systematic approach to transparency during development time is presented by Meis *et al.* [12]. Their research has mainly examined the flow of the personal data in a system in order to generate the static requirements of privacy that are related to personal information and its corresponding transparency requirements. Those requirements are static in the sense that they help the user understanding what data the system hold on them, but not necessarily about changes that may happen to this data, or how to execute their rights with respect to the data. In our research we develop a set of static and dynamic patterns that generate transparency requirements about user data. These patterns help to inform the user about their data rights under GDPR, who has access to their data, how it has been stored, *et cetera*. For instance, the user can be informed when data is collected, edited or accessed, as well as they are given the control on who accesses their data. Further to that, we focus on the application of the patterns in agile practices of development to help the requirements analyst and developers to generate and implement the resulting requirements.

3 Methodology and Patterns

To generate the transparency patterns we follow a *design science* methodology proposed for Information Systems [18]. This methodology 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 describe our approach to steps two to five. The first step is done through the review of the literature we present in Section 2, which identifies and justifies the problem we address. While the last refers to communicating the software artefact developed (*i.e.*, our patterns), we do so throughout this paper.

Definition of objectives. The regulation and literature advocate that users must have clarity on the data the system holds, the underlying mechanisms, data storage and access controls [15]. We define the objectives of our patterns in reference to these demands. We underline the pieces that led to such objectives.

The GDPR takes a user-centred approach. It regulates not only on what constitutes lawful processing of personal data, but on a series of rights that ensure data subjects (the end-users, or people whose data is collected and processed) are informed about such processing, and empowered to control it. In this work we focus on the right to transparent information, which states that the data controller (the system collecting and processing personal data) is responsible for presenting the data subjects with information that is transparent concise and clear[2]. This right is laid down by the regulation in reference to other Articles (such as Article 13 and 14, and 15 to 22) which define the content of the information to be presented to data subjects, including: the categories of data being processed[3], the purposes for processing it[4], and communication regarding their rights, such as rectification or erasure of personal data[5].

Existing literature reviews the demands of GDPR in view of systems development, often beyond the actual regulation and including information security into the list of concerns relevant to data subjects. For instance, they recommend that information should be shared with the data subjects regarding where data is stored, how data is protected and who has access to it [25], as well as information about the choices on limiting the processing of their data [13].

Design and development. The patterns are developed following a well-accepted methodology [28]. Requirements patterns generate specific types of requirements. We opt for them as they can be used during early stages of software developments and provide benefits such as reusability, consistent vocabulary and enhanced communication [11]. Additionally, patterns can be used to solve the issue of incomplete requirements [17]. Patterns are normally defined and classified in domain types. In our work, these domains are based on the system aspects (*i.e.*, data, operations) that systems are required to be transparent about. This

[2] *Ibid.*, Art. 12(1)
[5] *Ibid.*, Art. 19

paper only discusses Data Driven Patterns (DDP) [see full description of the patterns and their domains][6].

The patterns abstract requirements and systematically guide the analyst to address them early in the development. How to best present such information falls outside the scope of this work. For this topic, we refer to the literature on qualifiers of transparency [24] and user-friendly presentation of privacy-related information [20,16]. The patterns only help to decouple the presentation choices when passing information to the user by separating the contents from the design options (*i.e.*, whether users receive information promptly, or need to seek for it). This is done so that patterns can be applied by different teams of experts.

Demonstration. We applied our patterns in a case study with Spirit Healthcare, an organisation with a small technology branch who handles sensitive and individually identifiable medical data across several applications including remote patient monitoring, and education booking and management. Spirit Healthcare is one of 50 fastest growing companies in the UK. They are of manageable size and use SCRUM in their software development. The patterns are applied by their requirements analysts[7] and developers.

Evaluation. To evaluate the resulting artefact, we used a qualitative approach through semi-structured interviews, focus groups and questionnaires. The evaluation is conducted with the development team, and patterns run on an entire application where the domain experts and developers are the spokesperson for end users. We use thematic coding to analyse their responses.

3.1 Transparency Patterns

The patterns are described through templates that contain basic details (pattern number, last update, domain, and author); *applicability* information defining the cases in which to use the pattern; *contents* that need to be covered by the requirement; guidance on how to derive the contents; the *template* specifying how to write the requirement; and an *example*. Due to space constraints we show only essential parts of the template (*emphasised* in the text). Table 1 presents details on the *contents* and how to derive them.

Our patterns are data driven and include static and dynamic types. Static patterns describe information related to the data schema in a type-level (*i.e.*, data classes, attributes, and associations). While dynamic patterns cover the actual instance of data, *e.g.*, what data is currently stored about users and when it is being used. In Tables 2, 3, and 4 we present a summary of the Data Protection Transparency Pattern (static), the Data Subject Rights Pattern (dynamic,

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

[7] Even though the task of eliciting and analysing requirements can be assigned to different roles depending on the specific development process, in our context the distinction between such roles is not relevant. For example, in agile processes the product owner is responsible for requirements elicitation, but the scrum master and scrum team are involved in requirements analysis.

Table 1. Pattern contents and how the requirements analyst can define them.

Summary	Description
Data type	The system's data schema (classes, attributes, associations); also referred to as categories of data by the regulation (Art. 15(1)(b)).
Data instance	The actual contents of the data concerning the data subjects (*e.g.*, their specific *names*, or *addresses*).
Data interest	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* [13]).
Data storage	The conditions in which data is stored (*e.g.*, how data protected, if in an encrypted format, anonymous or pseudonymised, their retention period, location of storage [25], and others).
Data access	The recipients of data, either natural persons, or third party organisations (Art. 13(1)(e) and Art. 14(1)(e)).
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.
Data gathered	The source of the data (when not obtained from the data subject) and where the data has been gathered, inferred, or aggregated (Art. 14).
Conditions	Logical statements composed of combinations of data operations with the actors performing them, of the form «actor» «data operation» «data type», *e.g.*, "system admin updated address" where *system admin* is the actor, *updated* the data operation, and *address* the data type. These conditions are defined by the requirements analyst and are checked during run-time.

static), and Data Pattern (dynamic) respectively. A fourth pattern, the static version of the Data Pattern, is proposed but omitted due to space limitations. A complete list of the patterns can be seen in the supporting materials[8], this list also contains patterns developed for other domains not covered in this paper.

Table 2. Data Protection Transparency Pattern (with an example of its application).

Summary	Definition
Applicability	Use Pattern to generate transparency requirements for the specified data types of the data falling under data protection legislation the system holds about the end user. This pattern illustrates the data accessibility and storage.
Contents	Data interest. Data types. Data storage. Data access.
Template for *«data type»*	`If «Data Type» is a «Data Interest»`: The system must communicate to the user that it holds data of the type «Data Type» which has been «Data access» & «Data storage».
Example	
Template for *Address data*	The system must communicate to the user that it holds data of the type "address" which has been "accessed by system admin." This data is encrypted and will be stored until your account deletion.

3.2 Applying the patterns

To apply the patterns, the first step is to *identify data interests* based on user goals, laws and regulation. The requirements analyst uses this process to define data interests over the whole system for consistent results. Data interests are a reference to data that are defined under the same concern or information that matters to the user (*e.g.*, shipping data). This process gives flexibility to identify

[8] See footnote 6.

Table 3. Data Subject Rights Pattern (with an example of its application).

Summary	**Definition**
Applicability	Use this 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.
Information 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 «data subject's rights» can be performed by data subject.
Feature Template for *«data type»*	`If «data type» is a «data interest»:` In case *Condition* The system must implement action of «data subject's rights» on this data hold of «data type» that can be performed by data subject.
Example	
Information Template for *Address data*	The system must communicate to the user that it holds data of the type of *Address* and actions of *access*, *erasure*, and *data portability* can be performed by the user (data subject).
Feature Template for *Address data*	The system must implement an action of *access* on this data hold of *Address* that can be performed by data subject.

Table 4. Dynamic Data Pattern (with examples of its application).

Summary	**Definition**
Applicability	Use the pattern for the specified data instance that the system holds about the end user. This pattern gives information, during run-time, about what are the system actions being performed and by which actors.
Contents	Data instance. Data Interest. Data type. Data gathered. Condition.
Transparency of Instance of *«Data Type»* because *«condition»*.	`If «data type» is a «data interest»:` In case «condition », the system must communicate to the user « the «Data Type» (data instance) & «data gathered» by «actor» | «condition»»
Example	
Transparency of Instance of *"Address"* because *"system admin updated address"*	In case "system admin updated address", the system must communicate to the user "system admin updated address".
Transparency of Instance of *"Address"* because *"system admin updated address"*.	In case "system admin updated address", the system must communicate to the user "University of Leicester, University road, UK LX1RH" «gathered» by system admin.

any data group that relates to the user, not limited to personal data. Users' data concerns often depend on their goals, *e.g.*, "I want to see my shipping address before confirming payments". The analyst in this case creates a data interest called "shipping data" to represent this concern. For each data interest, the analyst can then derive the other concepts in Table 1.

Next, the analyst lists the user goals (given by user stories in agile development) that are (to be) implemented in the system. These goals are mapped to one or more of the pattern categories (Data, GDPR). Goals can refer to functionality already in the system, or to be added in the next iteration. Then, for each data interest, they identify the corresponding patterns and generate the requirements by following the pattern template. The last step is to finalise the transparency requirements document by applying a consistency check for removing duplicates and conflicted requirements, and aggregating requirements that can be combined. In particular, the analyst checks for unintended information disclosures or conflicts with security or privacy requirements. For example, information about data servers could cause a security breach.

4 Evaluation

The evaluation used qualitative approaches such as semi-structured interviews, focus groups, pre- and post-questionnaires. Participants included *domain experts* (DE1, DE2) with long clinical experience in the NHS, *senior developers*, and *developers* (D1..D4) from Spirit Health. Four pre-questionnaires were used to gauge the current agile development and GDPR practices in the digital team prior to the case study. Post-questionnaires captured how introduction of patterns impacted those practices and helped keeping users informed about their data.

The digital team manages their development life cycle by organising user stories into Product Backlog Items (PBIs) broken down into development tasks. The PBIs are arranged into biweekly sprints with free selection of PBIs by developers. PBIs are written primarily by the domain expert and senior developer and discussed biweekly in backlog reviews. Daily stand ups manage task progression, and communication with the business is through the product owner (domain expert) or the senior developer (also scrum master).

To analyse the data from questionnaires and semi-structured interviews we coded responses and clustered them to report results and find any similarities. An initial set of themes was extended in the coding processes, some derived from research questions (see Section 1) on how patterns impact on *time and effort of development*, and how *data and GDPR's transparency* are integrated into the process by the use of patterns. One theme was discovered while analysing responses: *informing the user*. The themes are discussed next.

Time and Effort. All participants agreed that agile requirements engineering practices reduce their workload. Moreover, participants D1, D2, D3 agreed and D4 neither agree nor disagree that they speed up the work in the project. After

applying the patterns to their user stories, the developers mentioned that the workload remains the same, if not increased. D3 mentioned this is "due to a more complete description of the tasks". Moreover, D3 mentioned that applying patterns will also speed up the work, in contrast with D4 who said that the work would not speed up "but it would improve the quality of the system".

Data and GDPR Transparency. The experts stated how important it is to report to the user what data is being collected. D3 reported that using the patterns helps to consider the relevant data interests and referred to an issue in their system: "We often collect data without telling the users what it is being used for, so it was good to consider 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". Having to systematically think of the data interests made developers and domain experts review the purpose for collection and processing data. As a consequence, where this purpose was no longer valid, they were able to apply data minimisation[9] (one of the pillar principles in the GDPR). The developers stated that the data interests help in grouping related data, then added "Data interests are a great extension on the concept of data tables – you can group many tables under a data interest".

Figure 1 presents an excerpt of an original user story and its requirements outputs after applying our GDPR transparency patterns. This illustrates how patterns systematically bridge gaps in the user story, *e.g.*, that *data is stored 2 years after account deletion* was not mentioned in the original user story. During a group session one developer pointed out the importance of informing the users about their data and access rights, D3 also mentioned "if people know their rights, they are more likely to fight for them".

Domain experts also stated that the patterns help building users' trust and enable them to make informed decisions. DE2 noted that it will help the user to decide if they want to use the software or not. Trust is equated with "a user feels very confident about the system" (quote from DE1, but also voiced by DE2). To achieve user confidence, the system needs to give "feedback [...] after carrying out commands" (DE1) and inform the users of their data (DE2).

Informing the User. D1 said "informing the user about the data being collected and the process being performed should be a requirement itself." D4 mentioned an example from their project experience on where and what they inform the user about: "We have a specific text on first login to our system (CliniTouch Vie) that the patient must agree to in order to use the system". It is clear that the team has considered transparency on a case-by-case basis, rather than systematically across the application. Previous practices have regarded transparency on a *need-to-know* rather than a *right-to-know* basis and not consistently applied it throughout the applications.

The developers have stated that the patterns help to sustain the user focus (the perspective fostered by the GDPR). D4 stated that applying the patterns

[9] GDPR, Art. 5(1)(c)

User Story: Edit personal details.

Test Case 1: When an email address is entered or changed an email requesting confirmation should be sent to the email address. The email should not be deemed valid until the link in the email has been clicked.

Test Case 2: When a user logs into the system, if their email address has been changed and has not been validated it should be stated that the email address has not been validated in the personal details and the user should be prompted at each login attempt to validate their email address. The username will remain as the previous email address before changes were made. A link should also be presented resending the email validation email.

Test Case 3: When any information is entered or edited in the personal details the old and new value should be audited in an audit table and timestamped along with the user who made the change.

Dynamic Data- requirement result:
If email address is updated by admin the system must communicate to the user, the currently in use email address which is being used by the user and other admins – informing them that the email could have been changed.

Data Protection-requirement result:
The system must communicate that it holds data of the type of personal details which has access by the admin team and selected members of other user types stored even after the account has been deleted due to auditing for at least 2 years.

Data Subject Rights- requirement result:
The system must communicate to the user that any changes made to their data are audited and details of the changer recorded.

Fig. 1. Original user story and its test cases, the GDPR resulting requirements.

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 patterns aid them to reflect on the data protection issues. One of the developers stated that a *user centred view* on transparency related mostly to the information disclosed directly by users. However, a big part of personal data is gathered indirectly via behavioural, statistical and usage data. This is often not identified as a data concern and the user is rarely informed about it. This data can be passively collected: "When the system collects information not directly from the user but could be critical – when attempted login – when successful etc...". For instance, IP addresses, information about time spent on the website, or any other behavioural data could affect the user or even be used to identify them. The data interests provide a way to discover all relevant data types and inform the user about them, whether they are collected directly or indirectly, as required by the rights described under GDPR Art. 13 and 14.

5 Discussion and Concluding Remarks

The patterns have been shown to systematically generate transparency requirements according to the GDPR. This transparency is extended to indirect collection of data where informing the user is often overlooked. This, in turn, increases

the developers' ability to structure actionable requirements in their applications. The patterns take in high level user stories, data interests and processes and generate transparency requirements on who has access to the data and the existing data processes. The patterns bring the users' perception to the implementation *by default* and drive the user focus (in line with the GDPR) to the point of writing code. Integrating transparency across the software process, the patterns are designed to fit agile practices and management tools. The dynamic aspect of patterns, using the concept of conditions, proves useful to build transparency contextually and in relation to user goals.

Transparency is a pillar concept in the GDPR, but our work's relevance is not limited to the European Union. There are international implementations of the GPDR that place similar emphasis on transparency, for instance the Data Protection Act in the United Kingdom[10], and the LGPD in Brazil[11]. Other prominent data protection regulations also touch upon the concept: the CCPA in California, USA[12] calls it the "right to know"; and the APPI in Japan[13] mentions the provision of information in its Articles 15 and 18, for instance. Although, a note of caution should be taken when discussing compliance with those regulations. Our patterns approach transparency from early development phases, but we do not claim they alone are sufficient for compliance with those regulations' requests. The GDPR, for instance, requires data controllers to provide the contact of the controller's Data Protection Officer. This request also composes transparency, but is not covered by our patterns, as it does not directly refer to personal data (our goal).

How personal data is handled in secondary processes (*e.g.*, collecting behavioural data) was all but completely ignored in the initial elicitation process carried out on the Spirit Health application. This information is not accessible through other standard elicitation methods due to not being directly related to the system's functionality, and is thus ignored. The patterns expose lack of transparency, particularly in secondary processes. The patterns generate details on what users must know about system aspects and data processing. This aids the requirements analysts' and developers' decision on what information to show. We believe this process should be formalised into a standard, so that it can be leveraged by other companies as well.

The case study with the Spirit Healthcare demonstrates that patterns fit well with agile practices. Current limitations in agile are that requirements are not well defined and the users' perspective is not fully included, because of a focus on rapid iteration. The patterns contribute to addressing these issues by extending the agile process. Teams can continue to work in a familiar structure whilst more deeply rooting the users' considerations into the process.

[10] `https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted`
[11] `http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709compilado.htm`
[12] `https://oag.ca.gov/privacy/ccpa`
[13] `https://www.ppc.go.jp/en/legal/`

The main limitation of our approach is that the completeness and consistency of the generated requirements highly depends on identifying the related data interests as one of the developers mentioned in the evaluation: "I can see the same things being repeated and to be able to standardise them across the application would be great". Therefore, it is important that the requirements analyst has sufficient domain and application knowledge. Another limitation is that transparency can overwhelm the user. Specifying different types of users (with different data interests) can help, and this option needs to be clear in the condition section. However, developers have not explored presentation choices in this evaluation.

Concluding remarks. The results from the evaluation showed that the patterns are effective in bringing the end-users' perception into the development process. This happens by informing users what data the system collects about them, how this data is used, and by whom it is accessed, according to GDPR requests. This information is provided in relation to the user goals. The patterns prompt the developers and requirements analyst to consider the data aspects that the user must know about. Furthermore, implementing the resulting transparency requirements has advanced other data protection concepts, such as exposing information which is not directly accessible to the user, thus ignored by standard elicitation methods, and the one of improving data minimisation through systematically reflecting on the purpose for data collection. This has been done systematically and based on the artefacts of agile development.

Future work for this research could investigate how to standardise the patterns using ISO standards. This could be used in a transparency certification awarded to companies that adhere to the defined practices by the standard.

Acknowledgement. The research is supported by University of Leicester. We also would like to thank Dr Mahmood Hosseini for the valuable input and Spirit Healthcare team for their collaboration, experience.

References

1. Cappelli, C., Leite, J.: Software transparency. Business and Information Systems Engineering **2**, 127–139 (2010)
2. Drury, M., Conboy, K., Power, K.: Obstacles to decision making in agile software development teams. Journal of Systems and Software **85**(6), 1239–1254 (2012)
3. Eberlein, A., Leite, J.: Agile requirements definition: A view from requirements engineering. In: Proc. of the Int. Workshop on Time-Constrained Requirements Engineering. pp. 4–8 (2002)
4. Erickson, J., Lyytinen, K., Siau, K.: Agile modeling, agile software development, and extreme programming: the state of research. Journal of Database Management (JDM) **16**(4), 88–100 (2005)
5. Herrnfeld, H.H.: Article 67 data protection by design and by default. In: European Public Prosecutor's Office. pp. 513–514. Nomos Verlagsgesellschaft mbH & Co. KG (2020)

6. Hoffmann, A., Söllner, M., Hoffmann, H., Leimeister, J.M.: Towards trust-based software requirement patterns. In: 2nd IEEE Int. Workshop on Requirements Patterns. pp. 7–11. IEEE (2012)
7. Hosseini, M., Shahri, A., Phalp, K., Ali, R.: Foundations for transparency requirements engineering. In: Int.l Working Conf. on Requirements Engineering: Foundation for Software Quality. pp. 225–231. Springer (2016)
8. Hosseini, M., Shahri, A., Phalp, K., Ali, R.: A modelling language for transparency requirements in business information systems. In: Int. Conf. on Advanced Information Systems Engineering. pp. 239–254. Springer (2016)
9. Kim, D.J., Ferrin, D.L., Rao, H.R.: A trust-based consumer decision-making model in electronic commerce: The role of trust, perceived risk, and their antecedents. Decision support systems **44**(2), 544–564 (2008)
10. Kizilcec, R.F.: How much information? Effects of transparency on trust in an algorithmic interface. In: Proc. of the 2016 CHI Conference on Human Factors in Computing Systems. pp. 2390–2395 (2016)
11. Loizides, F., Winckler, M., Chatterjee, U., Abdelnour-Nocera, J., Parmaxi, A.: Human Computer Interaction and Emerging Technologies: Workshop Proc. from the INTERACT 2019 Workshops. Cardiff University Press (2020)
12. Meis, R., Heisel, M.: Computer-aided identification and validation of privacy requirements. Information **7**(2), 28 (2016)
13. Meis, R., Wirtz, R., Heisel, M.: A taxonomy of requirements for the privacy goal transparency. In: Int. Conf. on Trust and Privacy in Digital Business. pp. 195–209. Springer (2015)
14. Moyano, F., Fernandez-Gago, C., Lopez, J.: Building trust and reputation in: A development framework for trust models implementation. In: Int. Workshop on Security and Trust Management. pp. 113–128. Springer (2012)
15. Murmann, P., Fischer-Hübner, S.: Tools for achieving usable ex post transparency: a survey. IEEE Access **5**, 22965–22991 (2017)
16. Murmann, P., Karegar, F.: 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)
17. Palomares Bonache, C.: Definition and use of software requirement patterns in requirements engineering activities. In: Proc. of REFSQ 2011 Workshops, REFSQ 2011 Empirical Track, and REFSQ 2014 Doctoral Symposium. pp. 60–66 (2014)
18. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. Journal of management information systems **24**(3), 45–77 (2007)
19. PRIVACY, G.M.: Consumer research insights and considerations for policymakers (2014)
20. Rossi, A., Lenzini, G.: Transparency by design in data-informed research: A collection of information design patterns. Computer Law & Security Review **37**, 105402 (2020)
21. Schwab, K., Marcus, A., Oyola, J., Hoffman, W., Luzi, M.: Personal data: The emergence of a new asset class. In: An Initiative of the World Economic Forum (2011)
22. Söllner, M., Hoffmann, A., Hoffmann, H., Leimeister, J.M.: How to use behavioral research insights on trust for HCI system design. In: CHI'12 Extended Abstracts on Human Factors in Computing Systems, pp. 1703–1708. ACM (2012)
23. Solutions, V.E.: Verizon 2014 data breach investigations report. verizon. com verizon. com (2016)

24. Spagnuelo, D., Bartolini, C., Lenzini, G.: Qualifying and measuring transparency: A medical data system case study. Computers & Security **91**, 101717 (2020)
25. Spagnuelo, D., Ferreira, A., Lenzini, G.: Transparency Enhancing Tools and the GDPR: Do They Match? In: Information Systems Security and Privacy. pp. 162–185. Springer International Publishing, Cham (2020)
26. Tu, Y.C., Tempero, E., Thomborson, C.: An experiment on the impact of transparency on the effectiveness of requirements documents. Empirical Software Engineering **21**(3), 1035–1066 (2016)
27. Turilli, M., Floridi, L.: The ethics of information transparency. Ethics and Information Technology **11**(2), 105–112 (2009)
28. Withall, S.: Software requirement patterns. Pearson Education (2007)
29. Zhu, K.: Information transparency in electronic marketplaces: Why data transparency may hinder the adoption of B2B exchanges. Electronic markets **12**(2), 92–99 (2002)