Value-Driven Analysis and Design: Applying Domain-Driven Practices in Ethical Software Engineering

Stefan Kapferer
Eastern Switzerland University of Applied Sciences (OST)
Switzerland
stefan.kapferer@ost.ch

Olaf Zimmermann
Eastern Switzerland University of Applied Sciences (OST)
Switzerland
olaf.zimmermann@ost.ch

Mirko Stocker
Eastern Switzerland University of Applied Sciences (OST)
Switzerland
mirko.stocker@ost.ch

Abstract

Business goals and economic values typically drive decisions in digitalization efforts and software development projects. There seems to be a lack of awareness that other values, such as ethical ones, should be respected to produce systems that do not harm human beings. Ethical values often are subjective; different stakeholders have different interests and priorities. So how can we make the values of all stakeholders of a software-intensive system transparent, especially the negative and positive impacts of the system on those stakeholders and their values, so that adequate decisions can be made and tradeoffs be found? The process pattern described in this paper suggests combining domain-driven analysis and design practices (modeling the domain, the software design, as well as its impacts and consequences) with value-based systems engineering (eliciting and prioritizing values and deriving value requirements from them) to make values and value requirements first-class citizens of the engineering process that go through the same refinement steps as business requirements and technical quality attributes and receive the same attention during design and implementation. We exemplify our process pattern in an online shop scenario that wants to add a same-day delivery feature, which has different ethical ramifications for several groups of system stakeholders.

CCS Concepts

• **Software and its engineering → Patterns**; **Requirements analysis**; **Software design tradeoffs**; • **Social and professional topics → Codes of ethics**.

Keywords

architectural decisions, domain modelling, ethical software engineering, requirements engineering, software design, value-driven systems engineering

ACM Reference Format:
Stefan Kapferer, Olaf Zimmermann, and Mirko Stocker. 2024. Value-Driven Analysis and Design: Applying Domain-Driven Practices in Ethical Software Engineering. In *29th European Conference on Pattern Languages of Programs, People, and Practices (EuroPLoP 2024), July 03–07, 2024, Irsee, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3698322.3698332

EuroPLoP 2024, July 03–07, 2024, Irsee, Germany
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1683-6/24/07
https://doi.org/10.1145/3698322.3698332

1 Introduction

Many recent developments in software and technology, such as artificial intelligence (AI) and robotics, have shown that systems that are released to the market not only have positive consequences on human and ethical values but negative ones as well. This is, however, not a recent phenomenon. The systems provided by the big tech companies that we all use on a daily basis have been reported to promote addictive behavior [4], make us lonely [24, 28], spy on us [6, 27] and deal with our stolen personal data [12, 20]. Codes of conduct, standards and guidelines towards ethical and value-based engineering exist [1, 15, 34] but do not always receive the amount of attention they deserve.

Software development is driven by requirements – functional and non-functional ones. Software engineers and architects need to know what the software should be able to do and which important qualities it should have. These requirements can include what the users should be able to achieve with the software. Still, they can also just focus on how the producers or operators of the software can achieve their business interests and economic goals. Often, functional requirements are based on the knowledge of the company or business experts who claim to know what the users and stakeholders need and want from the system. The Non-Functional Requirements (NFRs) usually focus on qualities such as *security*, *maintainability*, *scalability*, etc. that have to be respected in order to make the system work. When initiating software development projects, we have to consider whether digitalizing domains and processes is worth it; if the negative impacts of the system outweigh the positive ones, it might be in order not to go forward.

Development teams applying Domain-Driven Design (DDD) [14] and domain-driven requirements engineering practices, including Domain Storytelling [19] and Event Storming [8], aim to understand and model domain knowledge in order to find software architectures and designs that align well with the problem domain and take organizational structures into account (e.g., team topologies during development and operations). DDD focuses on communication between software engineers and business experts to establish that knowledge. A shared understanding of the problem and a so-called *ubiquitous language* shall be established. The idea of analyzing and modeling the business domain and then finding technically fitting software designs was already described by earlier Object-Oriented Analysis and Design (OOAD), e.g., [7]. Placing people and communication at the center of development processes was later promoted by the Manifesto for Agile Software Development as well.[1] We

[1] https://agilemanifesto.org

hypothesize that close communication between software developers and domain experts, promoted by agile and by domain-driven practices, is a sound starting point for discussions about ethical values and the impact of a system with all stakeholders.

To weigh the ethical pros and cons of a system and then develop it in a way that respects all stakeholders' values – or at least takes them into account and balances them with/against other requirements – developers must first become aware of the stakeholder values. To do so, we propose to combine value-based engineering with domain-driven modelling approaches and make values and the positive and negative impacts of a system transparent as a first-class concern during software development. The domain-driven modelling practices can help make important values transparent, force teams to discuss trade-offs, and make explicit digitalization decisions.

To work towards our vision and the proposal outlined above, we suggest a novel process called *Value-Driven Analysis and Design (VDAD)* in this paper. The VDAD process was developed as part of our *Just Enough Digitalization (JEDi)* research project, which is part of our broader research agenda towards a more responsible software engineering [21]. JEDi and VDAD target not only software engineers, but all human beings involved in the development of software. The results of the JEDi project including the process presented in this paper are also published as an Open-Source repository[2] and the *Value-Driven Analysis and Design (VDAD)* website.[3]

Note that we present new knowledge and research in this paper. While using the format of a pattern to describe the VDAD process, we can not identify any known uses yet (in the narrow sense of the term, following the rule of three).

The remainder of this paper is structured as follows. Section 2 discusses related work, focusing on ethics in software engineering and Domain-Driven Design (DDD). Section 3 presents our proposed process in pattern form. Section 4 applies the pattern to an online shop with "same day delivery" as a concrete example. It applies selected practices to illustrate how the pattern can be applied in practice. Finally, Section 5 summarizes the paper and outlines future work.

2 Background Information and Related Work

The IEEE Standard 7000 [1], which can be dowloaded for free, addresses ethical concerns during system design. Technical standards such as IEEE Standard 7000 cause additional effort for study and adoption; some of these standards are perceived as rather difficult to understand and heavyweight to use. Such entry barriers provide practitioners reasons to ignore the precious advice and guidance that could be found in the standard. The *ACM Code of Ethics and Professional Conduct*[4] provides a set of ethical principles and professional responsibilities for computing professionals; while this focus on individuals is laudable, a value-aware software design can not be expected to result from following such rules or advice alone. With the Ethical Software Engineering (ESE) repository[5] Zimmermann aims to combine the IEEE Standard 7000 with agile practices. ESE promotes a more lightweight approach gradually introducing ethical software engineering concepts to agile projects and teams [39].

According to Ozkaya [29], ethics qualify as software design concerns that should be treated as architecturally significant requirements. Value-Sensitive Design (VSD), as presented by Friedman et al. [17], is a theoretically grounded approach to technology design that accounts for human values. Winkler and Spiekermann [36] conducted a review of methodological practices in VSD projects. Spiekermann and Winkler [34] give a methodological overview of Value-Based Engineering (VBE) for ethics by design. Spiekermann published a series of further books on the topic [32, 33]. She splits VBE into three phases: *concept and context exploration*, *value exploration*, and *ethically aligned design*. The suggested workflow in the solution part of our pattern presented in this paper extends these ideas and integrates them into a domain-driven software engineering approach. We further detail the *ethically aligned design* phase as it is rather short and abstract/generic in [33].

Other approaches that aim at fostering ethical values in software are Ethical Design,[6] Inclusive Design [11] or Social Impact Assessment (SIA) [16]. Consequence Scanning[7] is an "agile practice for responsible innovators" that allows analyzing the potential consequences of a product or service on people. Alidoosti et al. [3] provides a comprehensive and systematic literature review on ethics in software engineering. They present an overview of investigations and activities in VSD methods, such as the identification of stakeholders, their values and value relationships. One of their findings is that only very little literature discusses the translation of values into software requirements.

More literature on ethics in software engineering exists. Often, it is specific to values within a certain problem domain. Very popular domains where ethics is already highly present these days are sustainability [25], big data and artificial intelligence [9], and privacy. The latter has been heavily addressed by regulations such as the General Data Protection Regulation (GDPR) in recent years [13]. However, our perception is that ethical thinking has still not reached mainstream software development. Human and ethical values are still largely ignored. With the VDAD approach that this paper proposes, we aim at showing how software can be engineered in a "value-driven" manner by combining ethical values with existing software engineering practices, methods and tools that are widely used in the industry. The approach intends to reduce entry barriers for software engineers and architects. Note that we do not discuss the term "value" itself in detail in this paper but refer the reader to the definitions of the IEEE Standard 7000 [1] and Value-Based Engineering [33].

The Domain-Driven Design (DDD) approach towards analysis, requirements, and software design, originally introduced by Evans [14], emphasizes the need for communication between project stakeholders. One goal of DDD is to overcome differences between technical and business people by analyzing the domain and establish a so-called "ubiquitous language". It is important to point out that DDD is focused on human activities, decisions and trade-offs; it aims at respecting all factors, not only technical, that influence decisions and application design. We believe that this mindset is

[2]https://github.com/ethical-se/value-driven-analysis-and-design
[3]https://ethical-se.github.io/value-driven-analysis-and-design
[4]https://www.acm.org/code-of-ethics
[5]https://github.com/ethical-se/ese-practices
[6]https://builtin.com/articles/ethical-design
[7]https://doteveryone.org.uk/project/consequence-scanning

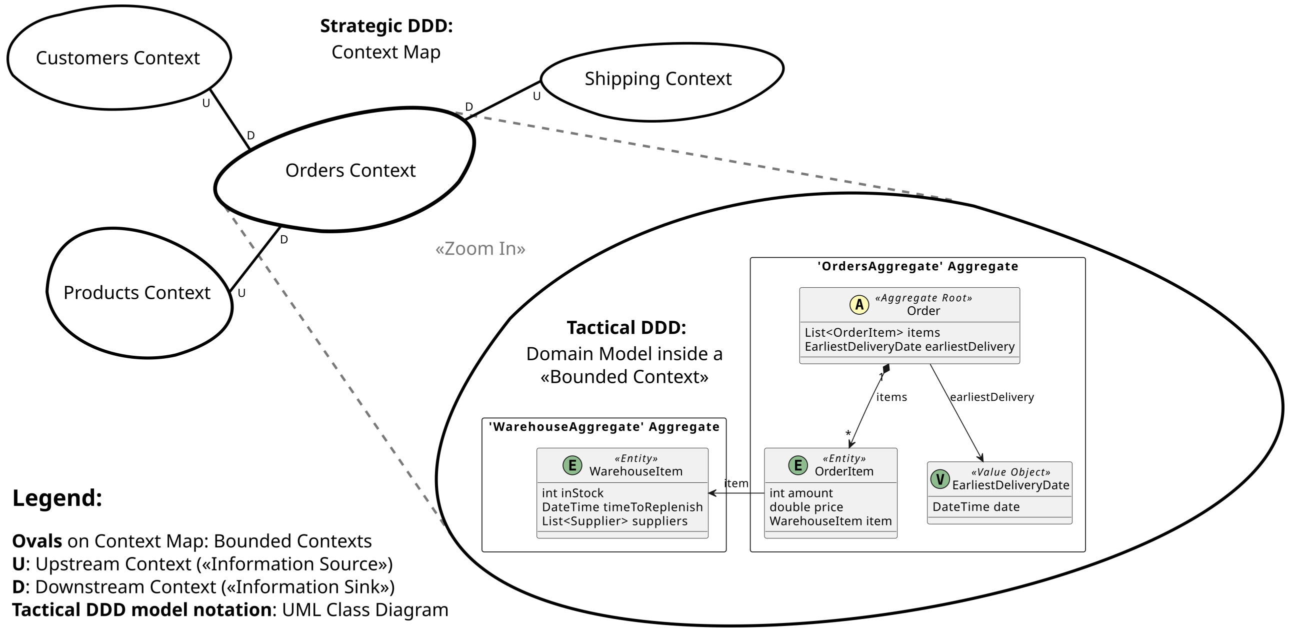

Figure 1: Domain-Driven Design (DDD) example: bounded context map and domain model zoom in (shopping/delivery scenario)

necessary to guide discussions about human and ethical values. We propose that while software engineers develop knowledge about their stakeholders and domains, they should also consider capturing the values of the stakeholders they communicate with.

By suggesting to apply DDD in our work presented in this paper, we do not limit this to software design and the original strategic and tactic DDD patterns [14] but domain-driven practices and modelling in a wider sense. Capturing the real world in a *domain model* and applying practices such as "event storming"[8] or "domain storytelling" [19] is mainly used for requirements engineering and object-oriented analysis (OOA) [26]. In DDD terms, analysis happens by modelling the *domain* and its *subdomains*. Domain models that visualize such subdomains and processes on an analysis level do not have to use the core DDD patterns already. They can be created in an initial customer workshop to model the gained understanding of the problem without concrete knowledge about DDD patterns. Models can be refined iteratively and substantiated towards implementation. These modelling practices that go beyond the original DDD patterns are commonly referred to as "collaborative modelling".[9]

Following an object-oriented design (OOD) approach [26], DDD divides a software system into so-called *bounded contexts* [14]. Every bounded context defines its own domain model that captures its core concepts and models them in terms of *aggregates, entities, domain events, and value objects*. Figure 1 illustrates an exemplary *context map* for an online shopping / delivery scenario – a diagram type used in DDD to show bounded contexts and their relationships. Spiekermann uses a similar diagram called "context diagram" as part of the "concept and context exploration" phase in [33]. Figure 1 further shows a "zoom-in" into one of the bounded contexts, the Orders context. Such a domain model captures the important concepts and their relations relevant for implementing a software system or component for that bounded context. Dividing the complete domain of a system into such bounded contexts makes it possible to discuss all kinds of design concerns – including values and value requirements – per context and, potentially, make separate digitalization decisions for each of these contexts. Solving human or ethical issues with a system may not imply that the whole system should not be built. Some contexts might be more critical than others. We suggest that the domain models of a system, be it on analysis levels with subdomains or on design level in terms of bounded contexts, can be enhanced to cover additional knowledge about stakeholders and their values. In Section 4, we will come back to this shopping/delivery scenario and apply our VDAD process to this example.

Modelling frameworks and tools for DDD, such as our *Context Mapper*[10] [22] can be used to support the process suggested in this paper. Note that one can implement the VDAD pattern presented in this paper with other software development practices and tools.

3 Pattern: Value-Driven Analysis and Design

also known as: *Domain-Driven and Value-Oriented Analysis and Design, Ethical Domain-Driven Design, Value-First Digitalization*

The pattern is a process pattern, similar to the Agile Architecture patterns published by Wirfs-Brock et al. [37] or the Scrum Pattern Group[11]. It shall be applicable by software engineers and teams aiming to provide digital solutions that respect human and ethical values. We describe the pattern in a slightly adapted EuroPLoP[12] format, splitting the *Problem* and *Forces* into two separate sections.

[8] https://ziobrando.blogspot.com/2013/11/introducing-event-storming.html
[9] https://archive.ph/Asn0C
[10] https://contextmapper.org
[11] https://www.scrumplop.org/scrum-plop
[12] https://www.europlop.net/patterns

Since we suggest a new method here, we do not discuss *known uses*. Please note that we consider the following process to be iterative and continuous. It should not end with the development of a system – new insights that can emerge during the whole lifecycle of the system might make changes necessary after initial development as well.

3.1 Context

An organization or project team wants to craft a new software system or enhance an existing system with a new feature. A product vision already exists. It is clear what the purpose of the product from a user's perspective is, and how the organization can benefit from the system on an economic level (business idea). Maybe first drafts of some functional requirements in the form of epics[13] or use cases already exist.

However, an analysis of all stakeholders and their values has not been conducted yet. It is not clear what the impact of the system on stakeholders and the whole society is. The system could potentially harm society as a whole or have a negative impact on its users or other stakeholders.

3.2 Problem

Industry software development practices often result in functional requirements that are primarily derived from business objectives. Although user-centric approaches such as Design Thinking [18], Lean Software Development [30], and Agile[14] exist, an empathetic perspective through the eyes of the users and all those affected by the system is often not considered or not adequately taken into account. In addition, these approaches often focus only on the users of a system and not on the invisible, less obvious stakeholders.[15] Methods that ask how a system will affect people such as Impact Mapping[16] exist, but are not always applied in everyday software development adequately.

As a consequence, software companies produce solutions that have not only positive but negative impacts on people's values. Once the software is on the market, users or other stakeholders may realize that their privacy is not respected, that the software is addictive, or that a digital tool has replaced valuable social interactions and made them lonely.[17]

Hence, the first problem to be addressed is the disclosure of values:

> *How to make the values of all stakeholders of a software-intensive system explicit? How to make the positive and negative impact of the system on these values transparent?*

The second set of questions progresses from analysis to realization:

> *How to derive software development requirements from the prioritized stakeholder values? How to address such value requirements during design, implementation, test, and maintenance?*

Finally, the third problem question connects value requirements with others:

> *How to deal with conflicts between value requirements, and between value requirements and other requirements (business, technical, regulatory)?*

3.3 Forces

The following forces stress the application of this pattern:

- **Goal conflicts.** Different users of the system or other stakeholders might have contradictory intentions. They might also have different values and use different value theories (for example *utilitarian ethics* vs. *virtue ethics* vs. *duty ethics* [1]). For example, a development team might want to conduct an extensive study to evaluate different user interaction patterns in order to identify an ideal solution for their users. Economic goals from the perspective of the company that finances the software, on the other hand, might suggest to just implement a pragmatic solution based on one's own experiences in order to fulfill financial goals such as reducing cost.
- **Value conflicts.** A system might negatively impact the values of some stakeholders while it has positive impacts for others. An example: An online video streaming platform analyzes its users' behavior (which videos were watched) and suggests new videos that might be interesting for the user by mobile notifications and emails on a daily basis. This has positive impacts on the operator of the platform since it increases the time the user spends on the platform. At first glance, the impacts on the user might also seem positive; through the suggestions, they might become aware of movies/videos they would not have seen otherwise. However, such systems aim at keeping the user hooked and promoting addictive behavior. Furthermore, such platforms can facilitate a "bubble effect", whereby users are drawn to similar videos on comparable topics. Seen in this way, they are dangerous to human health and the user's social life. Users might also not be aware that the platform is analyzing their behavior; some of them might not want to be observed and analyzed.
- **Dependencies between requirements and ambivalent consequences of decisions.** Value requirements lead to other requirements, and there might be conflicts. For example, a value requirement could be to protect the privacy of a user. With the term "privacy", we mean the value of privacy for a human being; not any technical measures or concrete non-functional requirements towards an implementation of a system. Such a value requirement might lead to non-functional requirements with respect to security measures or that certain data should not be persisted. Such data could be required to implement some functional requirements; data collection might be mandated by law. Note that some ethical

[13] https://www.agilealliance.org/glossary/epic
[14] https://www.agilealliance.org/agile101
[15] For example, in an online shopping scenario, developers might think about user experience and goals in a user-centric way, but not about the impact on other people, such as warehouse staff or delivery personnel.
[16] https://www.impactmapping.org
[17] Continuing the list of such examples of negative impacts of digitalized solutions would be straightforward. Such negative consequences are not necessarily intended – development teams do not always analyze all stakeholders and their values holistically as much as they should.

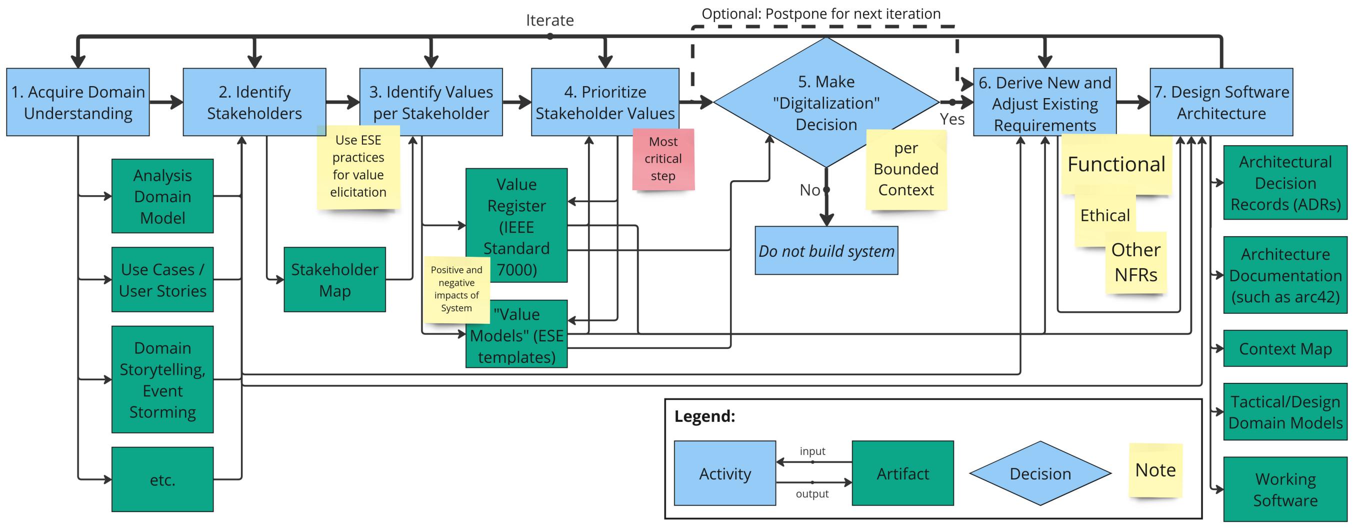
Iterate
Optional: Postpone for next iteration
1. Acquire Domain Understanding
2. Identify Stakeholders
3. Identify Values per Stakeholder
4. Prioritize Stakeholder Values
5. Make "Digitalization" Decision
Yes
No
6. Derive New and Adjust Existing Requirements
7. Design Software Architecture
Analysis Domain Model
Use Cases / User Stories
Domain Storytelling, Event Storming
etc.
Stakeholder Map
Use ESE practices for value elicitation
Value Register (IEEE Standard 7000)
Positive and negative impacts of System
"Value Models" (ESE templates)
Most critical step
per Bounded Context
Do not build system
Functional
Ethical
Other NFRs
Architectural Decision Records (ADRs)
Architecture Documentation (such as arc42)
Context Map
Tactical/Design Domain Models
Working Software
Legend:
Activity
input
output
Artifact
Decision
Note

Figure 2: Value-Driven Analysis and Design: An iterative process to produce software respecting human and ethical values, as well as economic and business goals, in a repeatable way

values, such as privacy, are already covered by regulations more than others, as discussed in Section 2.

- **Fuzzyness.** Values and value requirements are subjective by nature and, therefore, even more challenging to elicit in a specific and measurable way than other requirements.
- **Effort to elicit, analyze and design.** Ethical values are inherently subjective and often context-specific. They might change over time as the development team, users, and other stakeholders learn about a software system and its impacts. This makes them costly to elicit and respect; the system has to be kept up-to-date with the latest findings as well. Any effort spent on value and value requirement elicitation adds to the effort already required to understand and address business and technical requirements.
- **Uncertainty regarding long-term impact.** We often have to work with hypotheses to handle uncertainty. This is not only true for technical and architectural uncertainty [31], but for the unknown ethical impact of a system as well. Negative effects of a system, for example for society, may only become apparent after years of use. An iterative and continuous process is required to manage uncertainty and adjust products to latest insights.

Note that we consider the listed forces to be particularly important and relevant; an initial list resulted from a workshop conducted by the authors, which was later complemented with insights from discussions with early readers. That said, the list does not claim to be either mutually exclusive or collectively exhaustive.

3.4 Solution

The above forces cannot be resolved by a single, simple solution; their resolution on projects is highly context-dependent. Hence, we propose a value-centric analysis and design process rather than a one-size-fits-all design (or set of structural design patterns):

> *To elicit, negotiate, and address ethical values in software development, follow a Value-Driven Analysis and Design (VDAD) process – 1) understand the domain, 2) and 3) identify all stakeholders with their ethical values, 4) prioritize the values, 5) make conscious decisions about value-sensitive system development (starting with a go-no go decision), 6) derive and adjust system requirements, and 7) design the software architecture with stakeholder values in mind.*

The seven VDAD steps correspond to the activities (light blue boxes) in the VDAD workflow illustrated in Figure 2.

3.4.1 *Step 1: Aquire Domain Understanding:* This first step is covered already in the existing state of the art (i.e., existing methods), but still worth mentioning and including here:

> *Apply domain-driven practices to analyze the domain with its core, generic and supporting subdomains [14] and acquire knowledge about them. Establish a common vocabulary, the Ubiquitouous Language of the domain.*

Ideally, all stakeholders (from business experts to users and software engineers) are involved in this step. The domain knowledge of the experts and existing business processes are the *input* to this step. The knowledge is then written down and analyzed in depth (*output*):

- An analysis domain model,[18] maybe already per Subdomain, categorized into the three types core, generic and supporting [14] or even bounded context (but not necessarily in the first iteration).
- Workflows that illustrate business processes, elicited with domain-driven techniques such as event storming [8] or domain storytelling [19].
- Use cases and/or user stories, as summarized in [38].

[18] https://socadk.github.io/design-practice-repository/artifact-templates/DPR-DomainModel

- Already known Non-Functional Requirements (NFRs) and technical/organizational constraints. Examples of specific and measurable NFRs, e.g., regarding usability and supportability, can be found in the arc42 Quality Model.[19]
- Additional artifacts depending on project context and needs.

This step produces analysis results that describe the common understanding of the domain that has been established. These results do not exhibit design-level details (at least not in the first iterations). The domain model does not have to use tactic DDD patterns; it does not aim at being ready to be implemented yet. Event storming [8] or domain storytelling [19] can also be considered a viable alternative to domain modelling at this stage. The gained and established understanding must be captuted; in domain-driven terms, the real world – as it is – is described in this step.[20]

3.4.2 ***Step 2: Identify Stakeholders****:* Once a domain understanding is established in Step 1, it is possible to create an overview of all stakeholders; meaning all human beings that are somehow affected by the system that shall be developed. As discussed by Alidoosti et al. [3], a major challenge in this step is the identification of *invisible stakeholders* – stakeholders that do not directly interact with the system.

> *Recognize all (visible and invisible) stakeholders.*

Visible stakeholders are usually already identified when conducting functional requirements with user stories or use cases. Examples for potential invisible stakeholders are the environment and our society as a whole, the government, or other third party organizations that are impacted. Identifying invisible stakeholders requires anticipating who might be indirectly affected by the system. The stakeholder classification presented in [3] may help in this crucial step. Having a look at overarching core values [1] or crafting a “value map” [2] may uncover invisible stakeholders as well.

The output of Step 1 (domain model, user stories, and so on) serve as *input* to this step. Domain Storytelling [19] or Event Storming [8] results may also provide valuable hints here. The *output* of this step can be a Stakeholder Map[21] or a simple list of stakeholders.

Stakeholders can further be grouped; for example product managers and the board of directors might be subordinated to a stakeholder group called “management”. Additionally, there might be stakeholder groups that consist of a huge number of individuals. For example, the “users” group of an online shop or a social media platform is probably huge so that not every individual can be included in the process. In this case, representatives can be interviewed and/or the team could work with personas.[22]

3.4.3 ***Step 3: Identify Values per Stakeholder****:* In this important step, the values of each stakeholder group are elicited. Ideally, this is done in direct communication with the affected people. Alternatively, it takes an empathetic “putting yourself in the perspective of the people affected” approach.

> *Elicit the individual values of the identified stakeholders. Consolidate this information to create and then iteratively adjust a value register.*

This step takes the Stakeholder Map or list of the previous Step 2 as *input*. When performing this step, one can apply practices such as story valuation[23] and document the results in a form specified in IEEE Standard 7000 [1].

Output: We suggest that stakeholders and their values are modeled in the same way as domain knowledge. Value models could be created with DDD tools such as Context Mapper [23]. Values per stakeholder can be added to the *Value Register*, a concept introduced in the IEEE Standard 7000 [1]. The standard defines a *Value Register*: “An information store created for transparency and traceability reasons, which contains data and decisions gained in ethical values elicitation and prioritization and traceability into ethical value requirements.”

Once the project team is aware of how the system positively and negatively influences users and their values, domain models, use cases, user stories, etc., often have to be adjusted; VDAD is an iterative process.

When modeling all positive and negative impacts, one faces conflicts. Adjusting the system and the abovementioned artifacts probably cannot solve all of these conflicts. However, modelling these conflicts makes them explicit so that they are not overlooked.

3.4.4 ***Step 4: Prioritize Stakeholder Values****:* Once stakeholders and their values have been identified, these values can and should be analyzed in more depth regarding their importance and impact (a.k.a. consequences). This step in VDAD is both critical and challenging.

> *Prioritize the values identified so far. Identify the most important values for the team and where it is possible to compromise. Start and follow a requirements prioritization, conflict resolution, and tradeoff management process that involves all stakeholders.*

The output of Step 3, i.e., a value register that conforms to IEEE Standard 7000 or value statements such as those defined in Ethical Software Engineering (ESE), constitutes the *input* for this prioritization step. The *outputs* of this step are enhancements to the previously created value register and model (enhanced with *priorities* and *reasoning for tradeoffs*).

Some parts of the domain and system might be more critical than others. DDD divides systems in bounded contexts; hence, it can make sense to apply this step separately for each bounded context.

Support and advice for this important step can be found in ESE [39], Spiekermann [33], and IEEE Standard 7000 [1].

3.4.5 ***Step 5: Make Digitalization Decision****:* Systems should not be built if they are seen to cause more harm than benefit; teams should be aware of that. Digitalization decisions may be made for sub-systems or sub-processes.

Introducing this step and decision may sound unrealistic. One might ask: *Do the business leaders not overrule us anyway?* Well, first and foremost this step aims at making decisions conscious and explicit. If a project follows our suggested process, human and ethical values can no longer be “swept under the carpet” but one

[19] https://quality.arc42.org/qualities
[20] For the differentiation of these two perspectives (subdomains, OOA vs. bounded contexts, OOD), we refer the reader to the literature [14, 26, 35].
[21] https://miro.com/blog/stakeholder-mapping
[22] https://www.agilealliance.org/glossary/personas
[23] https://github.com/ethical-se/ese-practices

has to decide to ignore them actively. Whether and how ethical values are respected and upheld influences the choice of which company or organization to work for.

We also want to remind everyone involved in software projects that "being overruled" does not relieve them from responsibility. One of many real examples is the Volkswagen "Dieselgate" scandal, where in the end, not only the company faced consequences, but also software engineers were sentenced to prison [10]. One may consider the "Dieselgate" example as an extreme one; however, it illustrates ethical issues that arise in less severe situations and projects as well.

> *If the negative impacts of a part of a system seem to outweigh the positive values, decide within the team (with all stakeholders) whether it should be built or not.*

Input: The digitalization decision is made based on the values, priorities and tradeoffs found in Step 3 and Step 4. In DDD terms, it can be made for each bounded context.

Output: The minimal output of this step is a simple 'Yes' or 'No' answer to the "should we build this?" question. A more elaborate output includes a decision rationale, which can be found in the prioritized values from the previous step. The decision and its rationale can be captured as ADRs, structured according to one of the many templates that have been suggested in the software architecture community.

Note that the answer to the "should we build this?" question can also be "Yes, but …". It might be possible to find alternative solutions that allow a company to build a system without harming the identified values.

Additionally, note that we suggest to process the steps iteratively. Step 5 can also be postponed for a future iteration in case the team and/or stakeholders want(s) to delay the decision and gather additional insights upfront. The iterative approach also allows to revisit already made decisions.

3.4.6 *Step 6: Derive New and Adjust Existing Requirements*:
Once it is decided that a system shall be built, one has to adjust existing and derive new requirements from the produced artifacts that document stakeholders and their values. All outputs of the previous steps have to be respected as *input* in this step.

> *Treat ethical values as a type or category of Non-Functional Requirements (NFRs) that complement the desired software qualities. Aim at shaping the requirements in such a way that positive values are promoted and negative values contained as much as possible.*

This step does not only require deriving non-functional requirements from stakeholder values, but potentially also adjusting already known functional requirements. For negative values that cannot be argued away, one might ask: *How can we adjust the system to eliminate or minimize negative impact?*

The *output* of this step includes all kinds of requirements – functional, ethical, or other non-functional requirements. These requirements can be documented with plain text editors or word processors as well as state-of-the-art requirements management tools and/or special-purpose notations such as the ones that ESE suggests [39]. This step may also require conflict resolution, as new ethical requirements may conflict with existing functional or non-functional requirements.

3.4.7 *Step 7: Design Software Architecture*:
This step is business as usual, but still worth including and mentioning in VDAD.

> *Incorporate the values of your stakeholders in all decisions about architecture, coding, and testing. Apply domain-driven practices in order to apply the ubiquitous language about the domain knowledge, stakeholders and values to your software architecture, design and code.*

The *inputs* to this step and designing the software architecture are the requirements defined in Step 6. However, previous artifacts such as domain models or context maps influence the architecture as well when strategic and tactic domain-driven design are applied. All previous outputs should be taken into account. Analysis domain models that have already been created in previous steps might need to be detailed now by using tactical DDD patterns such as aggregate, entity, service and value object [14].

We refer to the following, exemplary activities of the Design Practice Repository (DPR) typically performed during this step:

- Architectural Decision Capturing[24]
- Architecture Modeling[25]
- Strategic Domain-Driven Design (DDD)[26]
- Tactic(al) design

The *output* of this step is indicated by its name, including software architecture artifacts such as Architectural Decision Records (ADRs),[27] component diagrams, and deployment models. arc42[28] suggests a twelve-part section structure to organize this design output.

The execution of the VDAD process over its seven steps can and should be iterative. This allows architectural prototyping, for example by implementing a Minimum Viable Product (MVP). Features can be added by applying vertical slicing; therefore this step is done in each iteration and your architecture as well as the related artifacts evolve incrementally.

3.4.8 *Process Continuation: Iterate over Steps 1 to 7*.
As indicated in Figure 2, neither the individual steps nor the entire process should be considered a linear, unidirectional sequence. Project teams typically iterate over each step and the entire workflow to keep values transparent and respect them during all software development lifecycle activities. While stakeholders of different kinds learn about domain context, requirements, and system capabilities, they might want to reprioritize and/or elicit additional values. Steps might also have bidirectional dependencies. For instance, while understanding the domain, new stakeholders might be found; an analysis of these stakeholders might bring new insights about the domain (Steps 1 and 2).

[24]https://socadk.github.io/design-practice-repository/activities/DPR-ArchitecturalDecisionCapturing
[25]https://socadk.github.io/design-practice-repository/activities/DPR-ArchitectureModeling
[26]https://socadk.github.io/design-practice-repository/activities/DPR-TacticDDD
[27]https://adr.github.io
[28]https://www.arc42.de

The process continues with state-of-the-art software engineering practices such as those collected by the Agile Alliance[29] to use the outputs of Step 7 and produce the mentioned working software, test it, validate and improve it iteratively – always re-evaluating that it respects the identified values. Ultimately and as early as possible, working software is produced; architecture models and code should always be in line with each other along the way.

We do not cover implementation, test, operations, and maintenance any further in our VDAD process. We consider these phases and their activities to be "business as usual" for the most part because value requirements are positioned as special types of NFRs both in IEEE Standard 7000 and in VDAD. Tools such as ArchUnit[30] and Context Mapper[31] can help keep the architecture/models in line with the actual code.

3.5 Consequences

The VDAD workflow aims to include an ethical perspective into the domain-driven development process and treat human and ethical values as first-class citizens.

3.5.1 Impact on Roles. An important value of Domain-Driven Design (DDD) is building shared knowledge and the "ubiquitous language". We recommend establishing the same common understanding about stakeholders and values. Therefore, it is necessary that Steps 1 to 4 are not assigned to specific project roles but rather done together with, ideally, all stakeholders in the project. Once an agreement on values and priorities has been reached, Steps 5 to 7 do not require every person to be involved. However, we would still recommend that business and domain experts, as well as technical people such as software architects are both involved here. The requirements and software architecture has, in the end, to be understood by the whole development team (per Bounded Context at least). However, related processes and practices such as the IEEE Standard 7000 [1] or ESE[32] discuss further useful roles, such as the "value lead".

3.5.2 Forces Resolution. The process pattern resolves (or helps resolve) the forces in the following way:

- **Goal conflicts.** Conflicts can be identified and discussed by making all stakeholders and their values transparent. The prioritization process and involvement of all stakeholders further help to find a good balance between the different goals and values.
- **Value conflicts.** Applying VDAD will not make the conflicts disappear but will help identify and discuss them to find acceptable compromises. VDAD steps 2 and 3 help to unveil conflicts and stimulate discussions to find tradeoffs.
- **Dependencies between requirements and ambivalent consequences of decisions.** Modelling brings traceability, which helps with dependency management. VDAD steps 4 and 6 help to unveil conflicts and stimulate discussions to find tradeoffs.
- **Fuzziness.** Processes can be iterated through so that uncertainty and misunderstanding are reduced along the way. Making ethical requirements explicit by writing them down in the same way as other requirements reduces uncertainty and ambiguity.
- **Effort to elicit, analyze, and design.** Introducing a process initially adds effort (learning, customization, usage). However, this effort usually pays off when applied pragmatically and consistently. Late work or rework is avoided by using a repeatable process that has a checklist effect.
- **Uncertainty regarding long-term impact.** Making known impact transparent often reveals other issues that would not have come up without elicitation. The VDAD process therefore helps to uncover potential long-term impact. However, the VDAD process does not yet cover evolution management after the development process. The resolution of this force may need to be improved in future work.

Additional patterns and other design elements and practices may also contribute to a further resolution of the forces (see related work section).

3.6 Process Summary and Comparison

Table 1 summarizes our suggestions of the previously described steps. It points out which domain-driven practices and artifacts we foresee in which steps, and why DDD, modelling and related domain-driven practices are important for our suggested process. However, as Table 1 indicates, some extensions to practices and artifacts are still required. We propose a new approach in the form of a pattern in this paper; therefore, there are no *known uses* (and no corresponding section).

Table 2 compares our own VDAD pattern with other approaches towards the problem such as the IEEE Standard 7000 [1], Value-Based Engineering (VBSE) by Spiekermann [33] and Ethical Software Engineering (ESE). The phases of the three compared approaches overlap. However, the compared approaches do not give much concrete advice with respect to our Steps 6 and 7 (requirements and design). One goal of our pattern and this paper is to assign the steps in the process their useful software practices, methods and tools. The last step of Table 2 (transparency management in the IEEE 7000 standard) is actually not covered by VDAD. Other than that, the different approaches all have their similarities.

4 Example: Online Shopping / Delivery

This section illustrates the application of our VDAD process to a concrete example.

Initial position: An online shopping provider is considering extending its platform and service with a "same day delivery" offering. There is a desire to guarantee same day delivery as a unique selling point, which is a logistical challenge. The management of the company is approaching the development team of the online platform to implement this epic. They want to discuss whether and how this could be implemented. Customers (shoppers), suppliers (of offered products), and partners (such as delivery firms) are other important stakeholders that have to become involved.

[29] https://www.agilealliance.org/agile101/subway-map-to-agile-practices
[30] https://www.archunit.org
[31] https://github.com/ContextMapper/context-mapper-archunit-extension
[32] https://github.com/ethical-se/ese-practices

VDAD Step	General Practices and Artifacts	Domain-Driven Practices and Artifacts
Step 1: *Aquire Domain Understanding*	e.g., use case modelling, user stories; business process models	collaborative modelling (e.g., event storming, domain Storytelling), analysis domain models
Step 2: *Identify Stakeholders*	in use cases and user stories; personas, stakeholder map	collaborative modelling (e.g., stakeholders as byproduct of Domain Storytelling)
Step 3: *Identify Values per Stakeholder*	stakeholder map, value register [1], ESE templates to model values	n/a (extensions required)
Step 4: *Prioritize Stakeholder Values*	Value Register [1], ESE templates	n/a (extensions required)
Step 5: *Make Digitalization Decision*	All inputs from steps before, existing decision records; making decisions based on modelled knowledge, respecting different stakeholders, deciding for trade-offs.	n/a (extensions required)
Step 6: *Derive New and Adjust Existing Requirements*	Use models and knowledge from previous steps to elicit and/or refine requirements, e.g., analysis domain Model	n/a (extensions required)
Step 7: *Design Software Architecture*	Architectural Decision Records (ADRs), architecture documentation (for example, filled-out arc42 template), working software	Strategic and tactic DDD (bounded contexts and design domain models using tactic DDD patterns) for architecture and design.

Table 1: Value-Driven Analysis and Design (VDAD) steps, eligible practices, and consumed/produces artifacts.

4.1 Step 1: Acquire Domain Understanding

In a first workshop with managers, developers and representatives of suppliers and delivery companies the main domain story from users perspective is discussed. Domain Storytelling [19] is used to gather knowledge about the story. Figure 3 illustrates a simplified, exemplary model that could be an output of such a workshop. The business idea is that if a customer faces an emergency situation and/or needs a convenience product as soon as possible, the online platform is able offer to deliver the product “on the same day” conveniently at a higher price.

Another output of this step, could be a domain model that covers important concepts (such as a “Same Day Delivery Availability” information that has to be calculated based on stock and customer address). In favor of the length of this paper we will not provide examples for all possible output artifacts.

In a next workshop the development teams and product owner start to fill the product backlog with items, which include an epic “same day delivery” and a number of INVEST-ready user stories.[33] The desire of the company to offer “same day delivery” serves as main driver for the epic and product vision.

The following five user stories illustrate examples that the team created in the backlog. They do not cover the problem completely but already illustrate that different stakeholders have different interests regarding this new epic. The stories are articulated in the human- and machine-readable notation Context Mapper DSL (CML).[34]

[33] INVEST stands for “I” ndependent (of all others), “N” egotiable (not a specific contract for features), “V” aluable (or vertical), “E” stimable (to a good approximation), “S” mall (so as to fit within an iteration), “T” estable (in principle, even if there isn’t a test for it yet). See https://www.agilealliance.org/glossary/invest.

[34] https://contextmapper.org/docs/user-requirements

```
UserStory SameDayDeliveryShopping {
  As an "Everday Consumer"
    I want to "order" a "Convenience Product"
    so that "I can respond to emergency situations
      conveniently without leaving home."
}
```

Story 2 is:

```
UserStory OrderDispatching {
  As a "Onlineshop Logistics Manager"
    I want to "automate and monitor" the "Supply Chain"
    so that "costs go down
      and customer satisfaction goes up
      (with as few items in stock as possible)."
}
```

Story 3 read as follows:

```
UserStory UserRegistration {
  As a "Product and Customer Relationship Manager"
    I want to "attract" the "Customers"
    so that "they shop with us
      and not with competition."
}
```

Story 4 takes the viewpoint of the delivery business partner:

```
UserStory OrderDeliveryInnovations {
  As a "Delivery Business Partner"
    I want to "offer" a "new services such as eCar
      delivery at affordable rates"
    so that "I win bids
      and business does not go to my competitors."
}
```

VDAD Pattern	IEEE Standard 7000 [1]	VBSE Book [33]	ESE Repository [39]
Steps 1 and 2	Clause 7: Concept of Operations (ConOps) and Context Exploration Process	Chapter 4, phase 1: concept and context exploration	n/a (business as usual)
Steps 3 and 4	Clause 8: Ethical Values Elicitation and Prioritization Process	Chapter 5, phase 2: value exploration	Story Valuation
Steps 5 and 6	Clause 9: Ethical Requirements Definition Process	Chapter 6, phase 3, first step: ethical value requirements	Story Valuation
Step 7	Clause 10: Ethical Risk-Based Design Process	Chapter 6, phase 3, subsequent steps: ethically aligned design	Definition of Ready, Definition of Done
n/a	Clause 11: Transparency Management Process	Chapter 7: transparency and information management	Ethical Review Meeting and Report, Value Retrospective

Table 2: VDAD steps compared with IEEE Standard 7000 and related work.

Finally, the exemplary story 5 introduces yet another type of stakeholder:

```
UserStory Regulator {
  As an "Auditor"
    I want to "review" the "online shop with its processes
      and use case realizations"
    so that "I can assess and assure their
      compliance with laws and good practices."
}
```

In addition to the functional requirements, the team elicited Non-Functional Requirements (NFRs).[35] The technical NFRs in the sample scenario include selected ones from the FURPS[36] classification scheme (with F for Functionality being covered by the above user stories):

- Reliability 1: Data accuracy and timeliness of data exchange is a high-priority requirement the shop system is supposed to satisfy; orders, shipments, and invoices as well as relations between these Domain Model elements, are of particular importance w.r.t. data accuracy and timeliness/freshness.
- Reliability 2: Web application and IT infrastructure security guidelines shall be adhered to protect the system as it processes the sensitive personal data of its stakeholders (see "As a" parts of user stories). Examples of such guidelines can be found at OWASP.[37]
- Reliability 3: The shop should be available 99.8% of the time, including weekends and public holidays.
- Performance: All requests should be responded to correctly within four seconds.

4.2 Step 2: Identify Stakeholders

With this step, the team aims to identify *all* roles and people in these roles that have an interest in the extended version of the online shop. Step 1 focussed on the *visible stakeholders*; with the help of [3], the team now creates a stakeholder map to further identify *invisible stakeholders*. Figure 4 shows an *initial* version of this stakeholder map (the team does not aim at making the map complete at this point; VDAD is applied iteratively and incrementally).

Stakeholders that are not directly visible from the user stories and domain storytelling in Step 1 are the government (that might give constraints through laws), other companies that provide the same offering, and staff of suppliers and delivery partners that might be affected.

4.3 Step 3: Identify Values per Stakeholder

Now that the team has identified stakeholders, the values and potential "benefits" and "harms" have to be identified per stakeholder.[38] Only a few examples are given that do not fully cover the analysis of all stakeholders that would be present in a real case.

Figure 5 illustrates a some of the key values identified by the team. Note that Figure 5 already contains priorities; these will be assigned in Step 4.

- **Customer / Shopper**: The customers and shoppers benefit from the new feature because they can solve their emergency situation; which might give them autonomy and freedom. On the other hand, this autonomy harms sustainability: The feature would lead to increased amount of trips that delivery partners have to do. In addition, the feature might not be sustainable in terms of the inner peace and patience of the customers. Our world is getting faster and faster anyway already, which increases our stress levels. The "Same Day Delivery" feature would further add to this phenomenon in society. As further indicated in Figure 5, there might be an ambivalent trade-off between "quality of life" and "privacy" for customers.
- **Logistics Team / Delivery Partner (Drivers)**: As shown in Figure 5, the logistics team and especially the drivers of delivery partners might be harmed in terms of "work-life balance". This feature will mainly increase work pressure on these stakeholders.

[35] https://ozimmer.ch/practices/2020/11/19/ExtraExtraReadAllboutIt.html
[36] https://en.wikipedia.org/wiki/FURPS
[37] https://owasp.org

[38] For a deeper discussion of what ethical values are, we refer the reader to IEEE 7000 Standard [1]. It also explains values such as "autonomy" and "care", both used in this example.

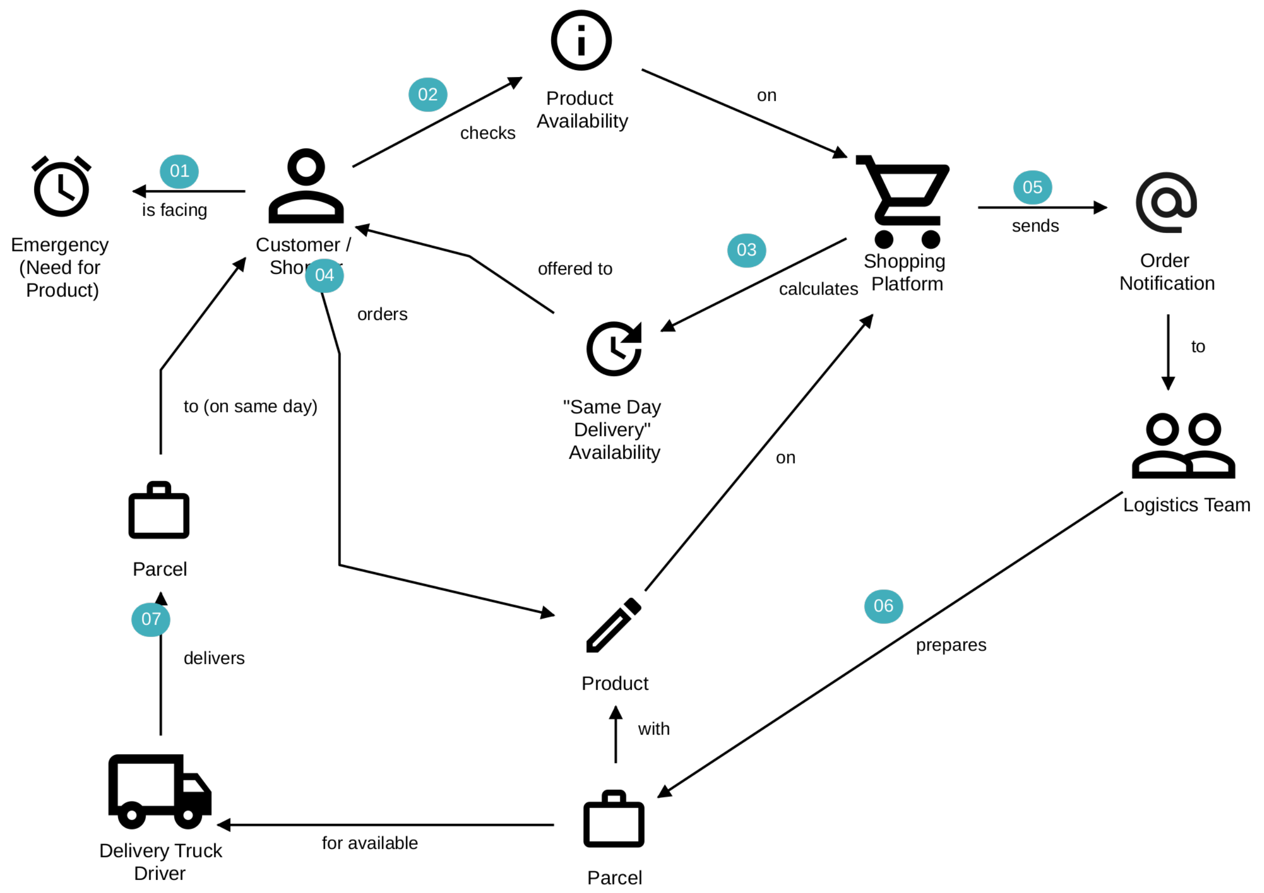

Figure 3: Sample domain storytelling for Same Day Delivery scenario

- **Shopping Platform Company**: The feature will increse company revenue, which is why the management welcomes and demands the feature.

This list and Figure 5 only provide a few examples. They are not complete, but it already demonstrates that there are conflicts between values and stakeholders. These will have to be addressed in the next steps.

The following examples show how values can be further documented with notations from the Ethical Software Engineering (ESE) repository that we introduced in Section 2.[39]

A sample value register entry is:

```
As a Delivery Business Partner,
  I value Care,
  as demonstrated in:
  - a realization of shop owner accountability to
     shareholders, investors, suppliers, and other
     stakeholders;
  - a reduction of responsiveness (and profit).
This value cluster has high (but not highest)
  priority for me.
```

The ESE notation used above is called *value epic*.

A second value register entry is:

```
When the online shop executes user story 1
(Online Shopping),
everday consumers/customers expect it to promote
and/or protect:
  - autonomy, independence, freedom and self-direction
    in particular
  - care, comfort and quality of life in particular
possibly harming
  - sustainability and respect (for delivery firm/staff)
by the following externally observable
  and/or internally auditable behavior:
  - increased hours of vehicles on the road
  - delivery team workload, overtime
```

Again, the notation is adapted from ESE. It is called *value narrative* there. ESE picks up concepts and wording from IEEE Standard 7000 such as value register and value cluster.[40]

4.4 Step 4: Prioritize Stakeholder Values

We already indicated in Figure 5 that the team started to prioritize values. This can be done by stakeholder, but in the end trade-offs have to be found in order to be able to make a digitalization decision for the complete epic. Some benefits or harms will always be

[39]https://github.com/ethical-se/ese-practices

[40]The ESE Glossary provides explanations of key terms in IEEE Standard 7000.

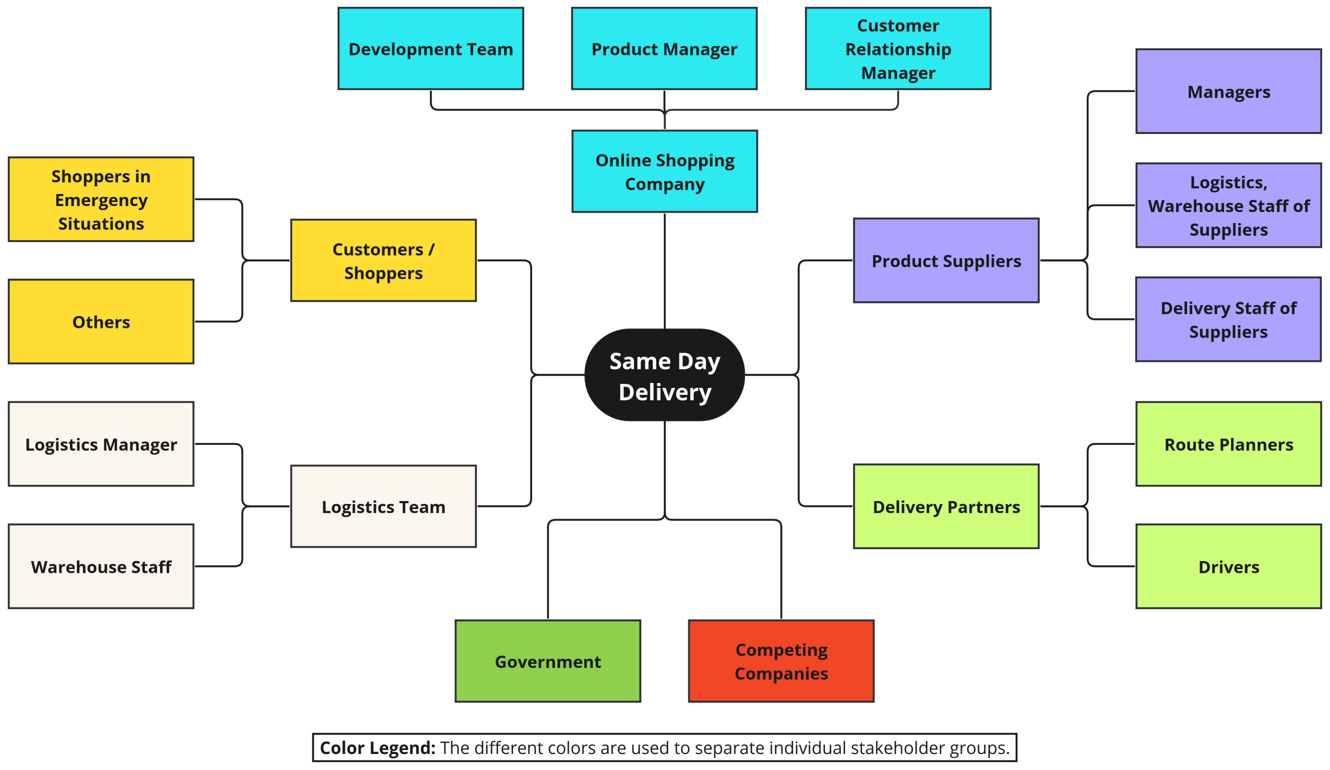

Figure 4: Sample stakeholder map for Same Day Delivery scenario

promoted, while for others, the team/company has to accept that they are harmed. A graphical representation (of parts of values as EVRs/VBSRs) as "value case diagram" is shown in the Figure 5.

In the following, we again use selected templates from the open-source ESE repository to show how such prioritizations can be documented. A sample value deals with user registration and prioritizes autonomy over sustainability:

```
UserStory CustomersAttraction {
  As a "Product and Customer Relationship Manager"
  I want to "attract" the "Customers"
  so that "they shop with us and not with competition"
    and that "business value, revenue and autonomy"
      are promoted
    accepting that "sustainability" is harmed
}
```

The notation used in the above example is an extended user story format written in the Context Mapper DSL (CML) language [23]. Since its Release 6.12, the Context Mapper tool supports this ESE notation as well as stakeholder and value modelling.[41]

Another value cluster and value register entry prioritizing efficiency over privacy (among other prioritizations) is:

```
In the context of Same Day Delivery epic for the
Online Shop (SOI),
the Onlineshop Logistics Manager (stakeholder)
values
  quality of life (of customers) and efficiency
more than
  privacy and
  quality of life (of delivery staff),
expecting benefits such as
  more profit,
running the risk of harms such as
  staff burnout.
```

This "value weighting" has been populated from core values in IEEE Standard 7000. Value weighting it is one of the formats that ESE suggests to produce lightweight but expressive, recognizable value register entries.

4.5 Step 5: Make Digitalization Decision

The team decides to implement the new user stories as the positive consequences outweigh the negative ones. The decision is formulated as a Y-statement:[42]

```
In the context of the same day delivery epic and its
user stories facing the need to be profitable,
but also wanting to respect the ethical values
of the stakeholders of the online shop,
the product owner decides to move forward
with the realization of the epic,
to achieve that
  the shop becomes even more attractive to its customers,
accepting that
  the pressure on delivery partners
  and delivery staff increases further.
```

[41] https://contextmapper.org/docs/vdad-support

[42] https://socadk.github.io/design-practice-repository/artifact-templates

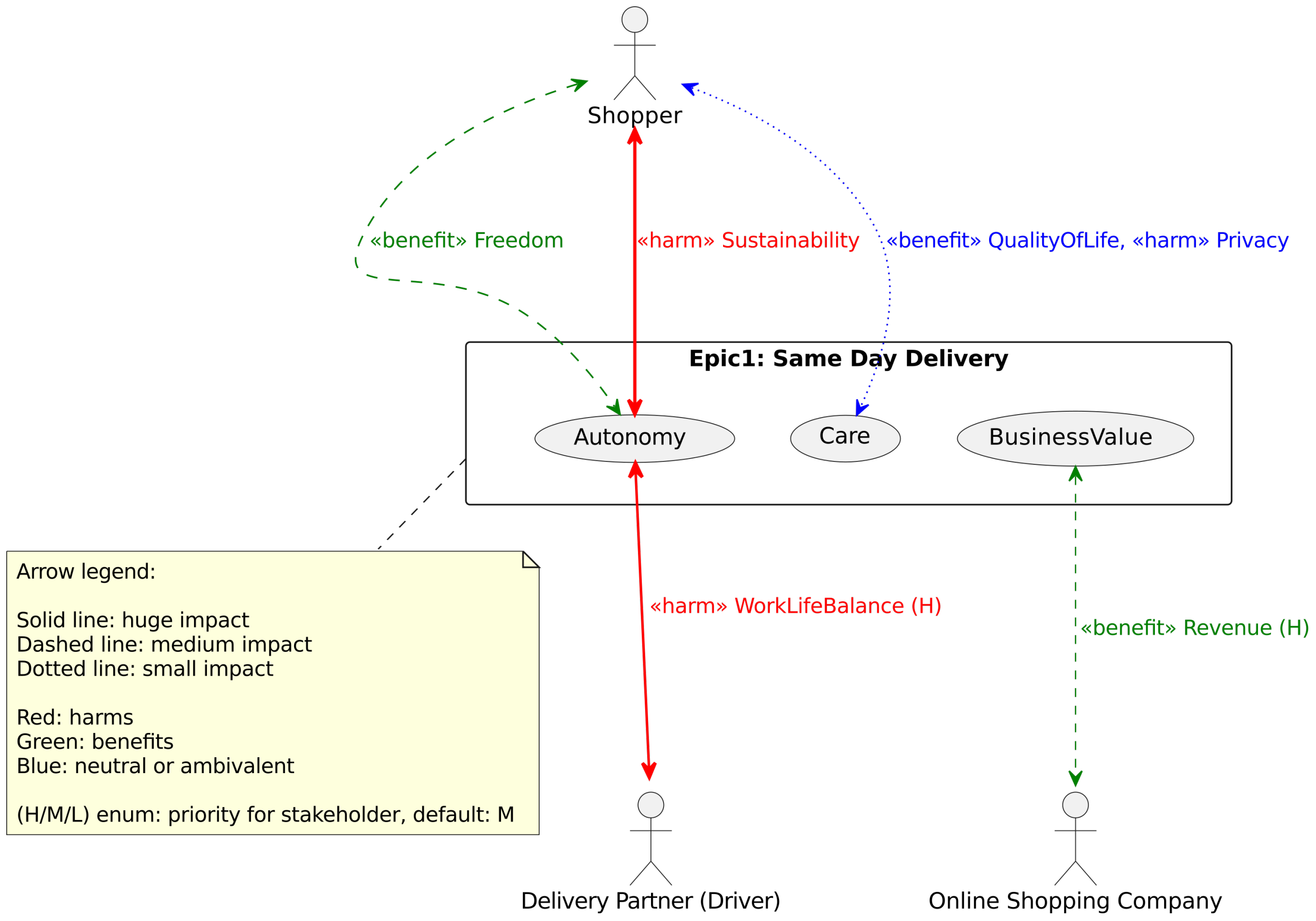
Shopper
«benefit» Freedom
«harm» Sustainability
«benefit» QualityOfLife, «harm» Privacy
Epic1: Same Day Delivery
Autonomy
Care
BusinessValue
Arrow legend:
Solid line: huge impact
Dashed line: medium impact
Dotted line: small impact
Red: harms
Green: benefits
Blue: neutral or ambivalent
(H/M/L) enum: priority for stakeholder, default: M
«harm» WorkLifeBalance (H)
«benefit» Revenue (H)
Delivery Partner (Driver)
Online Shopping Company

Figure 5: Sample value register for Same Day Delivery scenario

The team plans the following mitigation actions for the negative consequences of that decision:

- **Do not make *Same Day Delivery* the default**: The system should support this feature to win customers who need it in an emergency. On the other hand, the system should not create false incentives and make "same day delivery" the default. The feature shall be provided together with the corresponding wording "for emergencies". Thereby, the team wants to protect sustainability. If every order used the feature, sustainability would be harmed and delivery partners would not be able to handle it.
- **Pricing to set the right incentives**: The price for delivery must also be higher so that customers do not use the feature in every situation.
- **Availability based on staff workload**: To prevent staff burnout, the team foresees that the system should limit the availability of same day delivery based on the delivery teams workload. On days were the workload becomes "unmanageable", the system should react on the situation. This could mean that the "same day delivery" becomes unavailable for limited time windows or that the prices are adjusted based on workload. Strategies still need to be worked out in detail.

4.6 Step 6: Derive New and Adjust Existing Requirements

Based on the decision in Step 5, the team has to adjust existing requirements and derive new ones. For this example, we illustrate some selected sample Ethical Value Requirements (EVRs) first:

```
The system shall implement the user story "Same Day Delivery
  Shopping" without making it the default.
A customer should actively choose to be in an
  emergency situation and pay a fair price
  for the same day delivery service.
```

Another sample EVR is:

```
The system shall realize the user story "Order Dispatching"
  with 99 percent accuracy,
a made promise to deliver on the same day
  should only be broken in 1 out of 100 cases.
This is required to satisfy the value clusters from VDAD steps
3 and 4.
```

Finally, here are two examples of VBSRs, the first one addressing mitigation actions from Step 5 (in a format originally suggested in ESE):

```
To satisfy EVRs SameDayDelivery and OrderDispatching
and treat the risks related to it,
the high priority concerns of the Logistics Manager,
Delivery Partners and Shop Provider are:
  - the SOI has to implement technical risk treatment
    option Reliable Messaging
  - the system shall react on high workload of delivery
    staff and adjust availability of the same day
    delivery feature automatically
  - the Onlineshop Logistics Manager has to schedule
    for three shifts and find staff for them
  - the organization operating the shop has to pay for
    extra working hours of delivery staff
    and motivate staff to volunteer for unpopular shifts.
```

and, one in the form of a quality attribute scenario:

```
Stimulus: User Registration story
Concern Of: Product and Customer Relationship Manager
Observable When: Runtime, normal operations
Materializing In: User Interface, Business Logic Layer
  and EAI middleware in/of the SOI
Value Requirement: EVR-UserRegistration
Value Requirement Measure: 10% growth every quarter,
  values: inclusiveness, perfection
```

The quality attribute scenario format originates from “Software Architecture in Practice” [5]; it has has been adopted for our value-driven method context in ESE [39].

4.7 Step 7: Design Software Architecture

In this step, the SDD team applies more (strategic and tactic) DDD, makes architectural decisions and records them as ADRs. The resulting software architecture may also be modelled. See Section 3 for pointers to techniques and templates.

The following example ADR could be one consequence of the previous steps:

```
# Integrate Delivery Availabilities

## Context and Problem Statement
How should the shop system adjust the availability
of the same day delivery offer in order to satisfy
the EVRs defined in Step 6? How is it informed
that delivery can no longer be guaranteed on a
given day because the work-life of the logistics
and delivery stuff must be protected?

## Considered Options
* Manual action by the delivery staff
  (feature toggle)
* Integrate with planning/staffing systems of delivery
  partners for automatic detection of unmanageable
  workloads

## Decision Outcome:
Chosen option: Automated solution via system integration,
  because workload can be reduced earlier
  and additional manual tasks often lead to mistakes,
    e.g. when staff is very busy.
```

The ADR is written in the MADR format.[43] The Design Practice Reference/Repository (DPR) collects proven practices, including strategic and tactic DDD, as well as diagram types and notations[44] for architecture modeling and decision capturing. All activity and artifact descriptions in DPR come with additional examples [38].

4.8 Continue Iteratively

Our VDAD process pattern suggests to perform its steps iteratively. Step 7 is, therefore, not the end of the value-driven analysis and design work in practice. While working on design and architecture, the team gains new insights. The VDAD process can be restarted from the beginning to adjust related artifacts accordingly. All outputs mentioned in the individual steps should become more mature in every iteration of the process. Not every step has to receive the same amount of attention in every iteration.

5 Summary and Outlook

In this paper, we discussed how to make the ethical impact of a software-intensive system transparent so that the ethical values of the stakeholders become first-class concerns during software development; the positive and negative impact of the system on these values is made explicit so that it can be prioritized and conflicts be managed and mitigated. We presented a value-enhanced process called Value-Driven Analysis and Design (VDAD) in a pattern format. The seven steps of VDAD cover the software development lifecycle from analysis to design; they can and should not be performed sequentially but in an iterative and incremental way. As an important additional step, our process suggests making value-driven digitalization decisions explicitly for each system or subsystem (i.e., Bounded Context in domain-driven design terms).

As a direction for future work, one could also consider making the ethical and value-based design decisions and impacts of a system transparent to the end-user of the software product. By making decision records public, users could understand how decisions were made and why their creators may have accepted negative impacts. Furthermore, additional work is required in some of the presented steps to develop extensions to practices and tooling where needed.

The proposed process suggests a new approach; it is not a pattern that we have discovered and mined in projects in practice. We therefore plan to apply the process in projects to validate the approach in “real world” scenarios. This future work might lead to smaller patterns focussing on individual steps of the value-driven process proposed in this paper; it may also yield to pattern mining in the area of ethical software engineering.

We further plan to open our research for a broader target audience, as we hypothesize that in many cases the business plan and strategy of a company already has a major impact on ethical issues that will arise with the software they produce. The ideas of company founders, leaders, and sponsors presumably can not be “overruled” easily by software engineers. The problem we aim to tackle with our process, has to be addressed on the management level as well.

[43]https://adr.github.io/madr
[44]https://socadk.github.io/design-practice-repository/artifact-templates

Another open problem that we acknowledge is that many ethical and especially societal issues may only arise after years of experience with a new technology. For example, issues with social media platforms that we are aware of today (such as addictive behavior [4], issues with personal data [12, 20], etc.) were simply not known at the time they were developed – and the question remains whether a process like the one presented in this paper would have identified these problems at the time. In future research processes therefore might be extended to investigate how ethical problems with systems can be monitored over the long term.

Acknowledgments

We would like to thank our shepherd Uwe Zdun for his valuable comments and feedback, which improved the paper significantly. We also want to thank the participants of the EuroPLoP 2024 Writers' Workshop Berrin Akvardar, Monika Blattmeier, Veli-Pekka Eloranta, Allan Kelly, Ralf Laue, Daniel Lübke, Tsvetelina Plummer and Ville Reijonen for their valuable feedback.

References

[1] 2021. IEEE Standard Model Process for Addressing Ethical Concerns during System Design. *IEEE Std 7000-2021* (2021), 1–82. https://doi.org/10.1109/IEEESTD.2021.9536679

[2] Razieh Alidoosti, Martina De Sanctis, Ludovico Iovino, Patricia Lago, and Maryam Razavian. 2023. Stakeholder Inclusion and Value Diversity: An Evaluation Using an Access Control System. In *European Conference on Software Architecture*. Springer.

[3] Razieh Alidoosti, Patricia Lago, Maryam Razavian, and Antony Tang. 2022. *Ethics in Software Engineering: A Systematic Literature Review*.

[4] Adam Alter. 2017. *Irresistible: The Rise of Addictive Technology and the Business of Keeping Us Hooked*. Penguin Group.

[5] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice* (4th ed.). Addison-Wesley Professional.

[6] Zygmunt Bauman, Didier Bigo, Paulo Esteves, Elspeth Guild, Vivienne Jabri, David Lyon, and R. B. J. Walker. 2014. After Snowden: Rethinking the Impact of Surveillance. *International Political Sociology* 8, 2 (2014), 121–144. https://doi.org/10.1111/ips.12048

[7] Grady Booch. 1994. *Object-oriented Analysis and Design with Applications*. Benjamin/Cummings Publishing Company.

[8] Alberto Brandolini. 2021. *Introducing EventStorming - An act of Deliberate Collective Learning*. LeanPub. https://leanpub.com/introducing_eventstorming

[9] Adam Briggle, Katinka Waelbers, and Philip Brey. 2008. *Current Issues in Computing and Philosophy*. IOS Press.

[10] Ian Brooks, James Longhurst, Mario Kossmann, and Mohammed Odeh. 2018. Implementing the United Nations Sustainable Development Goals for the Systems Engineering of Multinational Corporations. In *INCOSE International Symposium*, Vol. 28. 1399–1411.

[11] P.J. Clarkson, R. Coleman, S. Keates, and C. Lebbon. 2013. *Inclusive Design: Design for the Whole Population*. Springer London.

[12] Greg Conti. 2008. *Googling security: how much does Google know about you?* Pearson Education.

[13] EU. 2016. General Data Protection Regulation. https://eur-lex.europa.eu/eli/reg/2016/679/oj

[14] Eric Evans. 2003. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.

[15] Association for Computing Machinery (ACM). 2018. ACM Code of Ethics and Professional Conduct. https://www.acm.org/code-of-ethics.

[16] William R Freudenburg. 1986. Social impact assessment. *Annual review of sociology* 12, 1 (1986), 451–478.

[17] Batya Friedman, Peter Kahn, Alan Borning, Ping Zhang, and Dennis Galletta. 2006. *Value Sensitive Design and Information Systems*. https://doi.org/10.1007/978-94-007-7844-3_4

[18] Hester Hilbrecht and Oliver Kempkens. 2013. *Design Thinking im Unternehmen – Herausforderung mit Mehrwert*. Springer Fachmedien Wiesbaden, 347–364. https://doi.org/10.1007/978-3-658-00371-5_18

[19] Stefan Hofer and Henning Schwentner. 2021. *Domain Storytelling: A Collaborative, Visual, and Agile Way to Build Domain-Driven Software*. Addison-Wesley Professional.

[20] Jim Isaak and Mina J. Hanna. 2018. User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection. *Computer* 51, 8 (2018), 56–59. https://doi.org/10.1109/MC.2018.3191268

[21] Stefan Kapferer, Mirko Stocker, and Olaf Zimmermann. 2024. Towards responsible software engineering: combining value-based processes, agile practices, and green metering. In *2024 IEEE International Symposium on Technology and Society (ISTAS)*. Accepted paper; to be published soon.

[22] Stefan Kapferer. and Olaf Zimmermann. 2020. Domain-specific Language and Tools for Strategic Domain-driven Design, Context Mapping and Bounded Context Modeling. In *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - MODELSWARD*. INSTICC, SciTePress, 299–306. https://doi.org/10.5220/0008910502990306

[23] Stefan Kapferer and Olaf Zimmermann. 2021. Domain-Driven Architecture Modeling and Rapid Prototyping with Context Mapper. In *Model-Driven Engineering and Software Development*, Slimane Hammoudi, Luís Ferreira Pires, and Bran Selić (Eds.). Springer International Publishing, Cham, 250–272.

[24] Jihyun Kim Kelly Merrill and Chad Collins. 2022. AI companions for lonely individuals and the role of social presence. *Communication Research Reports* 39, 2 (2022), 93–103. https://doi.org/10.1080/08824096.2022.2045929

[25] Patricia Lago, Roberto Verdecchia, Nelly Condori-Fernandez, Eko Rahmadian, Janina Sturm, Thijmen van Nijnanten, Rex Bosma, Christophe Debuysscher, and Paulo Ricardo. 2021. Designing for sustainability: lessons learned from four industrial projects. In *Advances and New Trends in Environmental Informatics: Digital Twins for Sustainability*. Springer, 3–18.

[26] Craig Larman. 2001. *Applying UML and pattern: an introduction to object oriented analysis and design and the unified process*.

[27] David Lyon. 2014. Surveillance, Snowden, and Big Data: Capacities, consequences, critique. *Big Data & Society* 1, 2 (2014). https://doi.org/10.1177/2053951714541861

[28] Martha Newson, Yi Zhao, Marwa El Zein, Justin Sulik, Guillaume Dezecache, Ophelia Deroy, and Bahar Tunçgenç. [n. d.]. Digital contact does not promote wellbeing, but face-to-face contact does: A cross-national survey during the COVID-19 pandemic. *New Media & Society* ([n. d.]). https://doi.org/10.1177/14614448211062164

[29] Ipek Ozkaya. 2019. Ethics Is a Software Design Concern. *IEEE Software* 36, 3 (2019), 4–8. https://doi.org/10.1109/MS.2019.2902592

[30] M. Poppendieck, T.D. Poppendieck, and T. Poppendieck. 2003. *Lean Software Development: An Agile Toolkit*. Addison-Wesley.

[31] Kelson Silva, Jorge Melegati, Xiaofeng Wang, Mauricio Ferreira, and Eduardo Guerra. 2024. Using Hypotheses to Manage Technical Uncertainty and Architecture Evolution in a Software Start-up. *IEEE Software* 41, 04 (jul 2024), 7–13. https://doi.org/10.1109/MS.2024.3383628

[32] Sarah Spiekermann. 2019. *Digitale Ethik: Ein Wertesystem für das 21. Jahrhundert*. Droemer.

[33] Sarah Spiekermann. 2023. *Value-Based Engineering*. De Gruyter, Berlin, Boston. https://doi.org/doi:10.1515/9783110793383

[34] Sarah Spiekermann and Till Winkler. 2020. Value-based Engineering for Ethics by Design. arXiv:2004.13676 [cs.CY]

[35] Vaughn Vernon. 2013. *Implementing Domain-Driven Design*. Addison-Wesley.

[36] Till Winkler and Sarah Spiekermann. 2021. Twenty years of value sensitive design: a review of methodological practices in VSD projects. *Ethics and Information Technology* 23, 1 (01 Mar 2021), 17–21. https://doi.org/10.1007/s10676-018-9476-2

[37] Rebecca Wirfs-Brock, Joseph Yoder, and Eduardo Guerra. 2015. Patterns to develop and evolve architecture during an agile software project. In *Proceedings of the 22nd Conference on Pattern Languages of Programs* (Pittsburgh, Pennsylvania) *(PLoP '15)*. The Hillside Group, USA, Article 9, 18 pages.

[38] Olaf Zimmermann and Mirko Stocker. 2021. *Design Practice Reference - Guides and Templates to Craft Quality Software in Style*. LeanPub. https://leanpub.com/dpr

[39] Olaf Zimmermann, Mirko Stocker, and Stefan Kapferer. 2024. Bringing Ethical Values into Agile Software Engineering. In *Smart Ethics in the Digital World: Proceedings of the ETHICOMP 2024. 21th International Conference on the Ethical and Social Impacts of ICT*. Universidad de La Rioja, Fundación Dialnet, 90–93. https://github.com/ethical-se/ese-practices