Philosophy & Technology (2021) 34:1085–1108
https://doi.org/10.1007/s13347-021-00451-w

RESEARCH ARTICLE

Ethics in the Software Development Process: from Codes of Conduct to Ethical Deliberation

Jan Gogoll[1] · Niina Zuber[1] · Severin Kacianka[2] · Timo Greger[3] · Alexander Pretschner[1,2] · Julian Nida-Rümelin[1,3]

Received: 4 November 2020 / Accepted: 26 March 2021 / Published online: 21 April 2021

Abstract
Software systems play an ever more important role in our lives and software engineers and their companies find themselves in a position where they are held responsible for ethical issues that may arise. In this paper, we try to disentangle ethical considerations that can be performed at the level of the software engineer from those that belong in the wider domain of business ethics. The handling of ethical problems that fall into the responsibility of the engineer has traditionally been addressed by the publication of Codes of Ethics and Conduct. We argue that these Codes are barely able to provide normative orientation in software development. The main contribution of this paper is, thus, to analyze the normative features of Codes of Ethics in software engineering and to explicate how their value-based approach might prevent their usefulness from a normative perspective. Codes of Conduct cannot replace ethical deliberation because they do not and cannot offer guidance because of their underdetermined nature. This lack of orientation, we argue, triggers reactive behavior such as "cherry-picking," "risk of indifference," "ex-post orientation," and the "desire to rely on gut feeling." In the light of this, we propose to implement ethical deliberation within software development teams as a way out.

Keywords Ethics · Codes of ethics · Codes of conduct · Software development

1 Introduction

Software systems play an increasingly important role in our lives. The public debate focuses in particular on systems that decide or support decision making on high-stake issues that affect third parties such as probation or creditworthiness

✉ Jan Gogoll
Jan.gogoll@bidt.digital

1 Bavarian Research Institute for Digital Transformation, Munich, Germany

2 Technical University of Munich, Munich, Germany

3 LMU Munich, Munich, Germany

(Eubanks, 2018; Noble, 2018; O`Neil, 2017). As our reliance on software-supported decisions increases, the demand for ethically sound software becomes more urgent. Software manufacturers find themselves in a position in which they are held responsible for unwanted outcomes and biases that are rooted in the use of software or its development process (or—in case of AI—the way the software "learned" (was trained) and the data that was selected for this training or "learning" process). In this article, we use the term software engineer broadly to include anyone who has a technical impact on the design of the product including programmers, database experts etc. working in an agile environment. In short, everyone involved and organized in agile teams during the development process. While it seems inappropriate and short sighted to shift responsibility entirely to developers, software companies still feel the need to address these issues and promote ethically informed development for two main reasons: Firstly, companies face backlash from unethical software in legal as well as in reputational terms. Secondly, companies and their employees have an intrinsic motivation to create better and ethically sound software, because it is the right thing to do.

In this paper, we will first briefly clarify the domain of the problem. Not every ethical challenge a software company faces should be dealt with at the software developer level (or the development team level). In fact, many possible ethical issues, for instance, the question if a specific software tool should be developed at all, fall into the wider domain of business ethics. After we have specified the opportunities and the domain of influence the software engineer actually has regarding the implementation of ethical values, we analyze one common approach to assist software engineers in their ethical decision making: Codes of Ethics and Codes of Conduct.

Here we will show why Codes of Ethics and Codes of Conduct (in the following the terms will be used interchangeably or simply referred to as CoCs) are insufficient to successfully guide software engineers (SE). We identify five shortcomings of CoCs that make them ill equipped to provide guidance to the engineer. Finally, we will argue that an approach built on an ethical deliberation of the software engineer may be a way to enable SEs to build "ethically sound" software.

2 The Responsibility of Ethical Decision Making in Software Companies

It is of crucial importance to define the domain, the scope, and the limit of ethical considerations that can be performed by software engineers and their respective teams before we can address the question of what ethical software development should and can do. Many issues that seem to be the result of software (and its development and use) are actually the result of certain business models and the underlying political, legal, and cultural conditions. Therefore, these challenges need to be addressed at the level of business ethics rather than within the development process of software. As an example, consider the implications for the housing and rent markets that stem from the adoption of services such as Airbnb. This paper is not so much concerned with these questions but with a somewhat narrower domain: After a business decision including ethical considerations has been made at management level, the development teams still have some leeway in deciding how to exactly develop the product. It is important to note that the amount of influence of management and development teams changes over time. While the former has exclusive decision making power in the early stages (for instance the decision whether a software should be created at all etc.), management has little control and influence in the development of a software product. The developers, experts in this very technical domain, develop the product within the given parameters. Naturally, these parameters will never be completely determined. Therefore, the development team has some leeway in the development of the product. An example is this: Imagine a care home facility where many elderly people do not drink enough water. A software engineer is tasked with implementing a technical solution that incentivizes drinking water. The setting is a smart care home and a smart cup is used to estimate how much water a person drinks in a day. Out of all the possible options and after some, we assume, sloppy deliberation, it was decided to link the cup to the smart TV and turn off the patient's TV, if they did not match their water quota. When asked why the developer chose that option, he answered that it met all technical requirements. Since he knows that the elderly love nothing more than their TV shows, they are sure to react to this.

While the feature in the example above has a clear impact on the ethical "side effects" of the overall system, it is also clear that many decisions will have no such impact. Whether a developer uses a *for* or a *while* loop to count to 100 makes no difference; in fact, both versions might result in exactly the same machine instructions. If a solution is recursive or iterative might impact performance, but will usually have no larger side effects. At the same time seemingly innocent design decisions can have a huge impact: choosing a binary datatype for a field in the database to store a person's gender reinforces this stereotype.[1] The way a software developer decides to store names might exclude most people on earth.[2] The layout and the technical design of websites might exclude certain groups from using them.[3] So while most decisions a software developer makes will have few, if any, ethical side effects, some decisions can have a disproportionate impact. Even if it is not possible to foresee all future uses and contexts of a piece of code, it is nonetheless crucial to integrate ethical deliberation at the base of the development process to turn ethical deliberation from a chore into a rational habit. Therefore, the development and incorporation

[1] For example, ISO 5218 specifies four values: "not known" (0), "male" (1), "female" (2), and "not applicable" (9), notably not offering a choice for non-binary persons. The data company Trifacta, for example, offers a dedicated "gender" data type; it can have two values: male and female (https://docs.trifacta.com/display/DP/Gender+Data+Type).

[2] For a list of common errors, see https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/. For a technical more technical example, just focus on Western European names, see: https://www.sqlservercentral.com/articles/proper-storage-and-handling-of-personal-names

[3] For design guidelines for visually impaired persons, see: https://fuzzymath.com/blog/improve-accessibility-for-visually-impaired-users/. For the problem that javascript can cause screen readers, see: https://medium.com/@matuzo/writing-javascript-with-accessibility-in-mind-a1f6a5f467b9

of ethical principles in software engineering enables the combination of proactive technology ethics[4] with normative ethics. This highlights how real-life actions and semantic contexts are mediated and transformed by the deployment of software (systems) (c.f. Reijers & Coeckelbergh, 2020; Vallor, 2016). Such an approach to software engineering requires empowered individuals on the one hand and proper structural conditions that allow for such a normative method to flourish on the other. Because agile processes (at least in theory and if implemented correctly) foster individual empowerment while providing structural empowerment, we focus on ethical deliberation embedded in them. It is clear that not all software is developed in an agile manner. Yet, a large proportion of companies have implemented or are planning to implement agile development according to surveys (Digital.ai, 2020; Hewlett, 2017; Koning & Koot, 2019). In fact, the popularity of agile methodologies has constantly increased over the last decade. It is often argued that "empowerment and involvement in decision making are often seen as core strengths of agile" (Drury et al., 2012) and only when the developer feels empowered to make his or her own decisions about the design and the implementation of a product, normative deliberations can be meaningfully employed at the level of the individual developer. Thus, agile processes can enable ethical deliberation by providing a structure that enables ethical deliberations, because the sharing of information between team members is explicitly encouraged and individuals interact constantly and on a daily basis.

The question then is as follows: How should software engineers approach ethical questions and what tools may facilitate ethical considerations? Figure 1 provides an overview of the different levels of ethical decision making in a company. This graph is obviously a stark simplification of reality, but it helps to illustrate the fact that many decisions have already been made before tasks are assigned to developers.

Consider the following example: Technological progress enables us to build a robot that offers support to the elderly which includes a potentially wide variety of tasks that may cover the entire field of geriatric care. Figure 1 illustrates the different layers of ethical decision making (pertaining to manufacturing the robot). Every organization is embedded into a web of social expectations, legal requirements, and cultural norms. In our example, it is politics and a societal discourse that establish if nursing robots are desirable at all. The eventual consensus is influenced by developments such as demographic transition in developed nations, the political goal of providing care for senior citizens, and the capacity of the healthcare system. Once it has been established that it is legal to build a specific product or artifact and a societal consensus has more or less been reached, the technology may be tested (and introduced) and the management of a company can then decide to manufacture the product (for an extreme example, consider the weapon industry). Next, a project team within the company will

[4] We understand the term proactive information ethics, as a normative procedure taking place continuously within the design process and thus influencing the shape of the artifact. Different concepts were developed to address such an approach to technology ranging from Brey's anticipatory technology ethics, Floridi's constructions approach, Palm and Hansons ethical technology assessment, or Boenik et al. just to mention a few. In this paper, we cannot go into highlighting differences or similarities of their concepts and the integration into our ethical method (see for example Moor, 1985; Floridi, 1999, 2013 esp. Chapter 8; Verbeek, 2005, 2011; Brey, 2011, 2012; Palm and Hansson, 2006; Boenink et al., 2010).

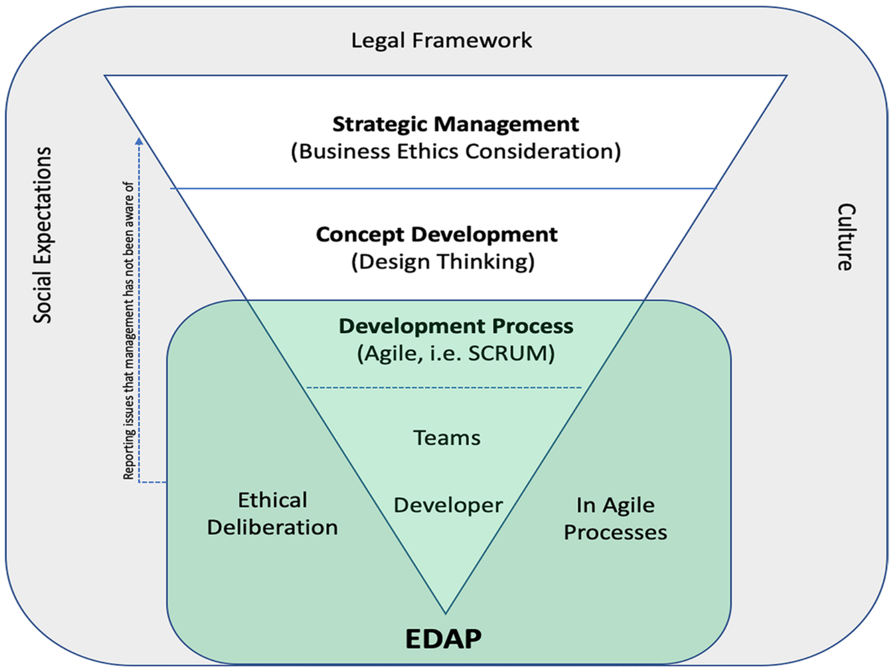

Fig. 1 The domain of EDAP and the different responsibilities

decide on the exact specifications of the product. The nursing robot's design can focus on different geriatric aspects, e.g., to only assist human care workers, to be fully autonomous in its care for an elderly human, or to be deployed in specific places only such as hospitals where the robot's activity is subject to strict limitations and under constant human supervision. This initial requirements elicitation phase already addresses and decides several ethical questions. It is then only within the narrow confines of these specifications that a developer can and should influence the product's design. Developers can, for example, choose a technology that protects the user's privacy but still ensures the achievement of established business objectives. Our care robot might use video to interact with a patient, but it is perhaps possible to store the data in a privacy-preserving way, or even to design the robot in such a way that the data does not need to be stored at all. Even small components of the robot can have disproportionate ethical side effects. For example, if the robot can only be named using ANSI characters, it could not have a native name in most of the world. If the robot's software is built in such a way that it can only have a male and a female personality, it might make extending it to a third gender prohibitively expensive. Of course, if the software engineer realizes that a major higher-level ethical issue might have been overlooked and not taken into consideration, she has a duty to clarify and check whether the issue has indeed been overlooked. Table 1 provides an overview of different ethical questions as well as which actors make decisions and where ethical deliberation should be located.

Table 1 The different levels of ethical issues in the software engineering process

Domain	Issue	Output	Actors
1. Politics	Should the elderly be taken care of by robots?	Legal framework, societal and cultural conditions	Society
2. Strategy/business ethics; corporate social responsibility; corporate digital responsibility	Should *we* build a robot that supports and cares for the elderly? What is the business model?	Project (approved)	Company/institution
3. Product conceptualization (e.g., design thinking)	What capability should the robot possess?	(concrete) Requirements	Project team, subdivision etc
4. Development process (e.g., scrum)	How do we implement concrete features considering the given (above) parameters?	Product	development team

Once we reach the phase of development, the decision to build the robot has already been made, the business model has been chosen and specific demands have been outlined. Any remaining ethical questions must be dealt with by the software engineer or the development team. Of course, there are differences between companies and corporate culture which in turn influences the degree of management's involvement and to what extent it fosters ethical decision making at the development level. Yet, the developer usually has the greatest influence in translating ethical considerations into the product, when it comes to the implementation of the pre-defined parameters into software. If, as should be the case in agile organizations, teams are given high-level problems, e.g., "Find a way to keep birds off some property," an ethical deliberation at the engineer level will help to make explicit different options and weigh them in terms of ethical considerations, such as:

- A team member with a background in construction might suggest using spikes on rims and poles where birds like to sit.
- One with a background as a falconer might suggest getting a falcon to scare off other birds.
- A sound engineer might suggest using high-pitched noises.
- An environmental activist will suggest catching and relocating the birds.

This example illustrates the possibilities and the sphere of influence of the engineer. It cannot and should not be the case that the engineer in this example has to decide whether it is ethically justifiable to limit the movement of birds *at all*. Rather, given the constraints set at higher levels and through decisions made earlier in the process, the engineer should focus on ethical considerations that are in her domain and where she can assert influence.

Now that we have established the domain in which software engineers have "ethical" influence over an outcome, the question is as follows: How do we enable engineers to build software ethically and how do we adequately consider potential ethical issues and find solutions to these questions?

We have to acknowledge the fact that software engineers are usually not specifically educated in ethics and have not had intensive training or other experience in this domain. A prominent method to address the mismatch between the lack of ethical training and the impact a product might have and, therefore, the ethical attention it should receive has been the publication of Codes of Ethics and Codes of Conduct. In the following chapter, we argue that this approach is ill equipped to achieve its intended purpose of being a useful guideline for software engineers.

3 Codes of Conduct and Codes of Ethics

Codes of Ethics and Codes of Conduct have been published in order to give ethical guidance to engineers and management.

CoCs, for instance, published by institutions such as IEEE (Institute of Electrical and Electronics Engineers) and the ACM (Association of Computer Machinery), supranational institutions such as the EU High-Level Expert Group on AI and UNDP, or the tech industry (Whittlestone et al., 2019), have a central, (self-)regulatory function in the discourse on the development of ethically appropriate software systems. They represent a more or less sufficiently complete and mature surrogate of various normative positions, values, or declarations of intent, which ought to be implemented in an adequate form in the process of software development.

The main contribution of this paper is to analyze the normative features of CoCs in software engineering and to explicate how their value-based approach might prevent their usefulness from a normative perspective. To this end, we identify the most prominent kinds of values and principles in these codes, what kind of normative guidance they can provide and what problems might arise from those CoCs that uphold a plethora of abstract values.

3.1 Codes of Conduct, Values, and Principles in AI and Software Engineering

Codes of Ethics (CoEs) or Codes of Conduct (CoCs) are intended to provide guidance to engineers who face ethically relevant issues and provide them with an overview of desirable values and principles. The ACM Code of Ethics and Professional Conduct—for example—declares that:

> "Computing professionals' actions change the world. To act responsibly, they should reflect upon the wider impacts of their work, consistently supporting the public good. The ACM Code of Ethics and Professional Conduct ("the Code") expresses the conscience of the profession.
> The Code is designed to inspire and guide the ethical conduct of all computing professionals, including current and aspiring practitioners, instructors, students, influencers, and anyone who uses computing technology in an impactful way" (Gotterbarn et al., 2018).

Furthermore, the ACM Code demands that computer professionals act in accordance with their general principles. The normative character of rules in the code suggests that engineers should behave as indicated and are judged accordingly. Partly, they are designed as self-commitment but also include punishable legally binding obligations. Of course, CoCs address a specific professional area and, therefore, remain specific in their formulation of certain values. Nevertheless, when we consider the nominative function of codes, the specification loses urgency, i.e., when we consider the normative requirements that codes should fulfill. We find the same normative requirements across all industries: "They are guiding principles designed to maintain values that inspire trust, confidence and integrity in the discharge of public services" (Secretariat Treasury Board, 2003).

There is a plethora of CoCs that address software engineers in general and developers working with artificial intelligence in particular. Initially, CoCs were introduced by businesses as a response to increasing problems of corruption and misbehavior in business practices. Over the years, the adoption of CoCs has spread to many other domains, especially engineering and medicine, but also business. Davis (1998) has argued that "a code of professional ethics is central to advising individual engineers how to conduct themselves, to judging their conduct, and ultimately to understanding engineering as a profession." CoCs would thus serve three main purposes: Firstly, they guide the individual engineer and help to avoid misbehavior. Secondly, they serve as a benchmark for other actors in a profession to judge potential behavior as unethical and thereby contributing to the reputation of the profession as a whole. Finally, they help to define the self-image of a profession by setting a rulebook for what a professional actor should or should not do—this might be particularly relevant for a comparatively young field such as software engineering. Schwartz (2001) outlines eight metaphors that describe how individuals may interpret CoCs: as a rulebook, a signpost, a mirror, a magnifying glass, a shield, a smoke detector, a fire alarm, or a club (ibid.).

Questions of the CoCs' effectiveness, however, have been raised early on. Schwartz (2001) conducted 57 interviews in the domain of business ethics and reported that less than half of the codes actually influence behavior. Kaptein and Schwartz (2008) find mixed results regarding the relationship between CoCs and corporate social responsibility performance of companies. More recently and specifically examining CoCs for software engineers, McNamara et al. (2018) have conducted a vignette experiment to test the influence of CoCs on developers. They found no correlation, stating that "explicitly instructing participants to consider the ACM code of ethics in their decision making had no observed effect when compared with a control group" (ibid.). While the jury is still out on the empirical effectiveness of CoCs, this paper is not overly concerned with this issue. Rather, we attempt to show that CoCs conceptually fail in various ways when it comes to their main goal: providing ethical guidance to software engineers who find themselves in uncertain situations.

Some research has been conducted that compares ethical codes and their values and tries to quantify them with the goal of establishing a potential consensus. The main focus of the current literature has been on CoCs that deal with the development of artificial intelligence systems.

Jobin et al. (2019), for instance, coded 84 documents within the domain of AI CoCs and summarized eleven main principles (ordered according to the number of documents that contain the principle, descending): transparency, justice/fairness, non-maleficence, responsibility, privacy, beneficence, freedom/autonomy, trust, sustainability, dignity, and solidarity. With "transparency" mentioned in 73 out of 84 (87%) to "solidarity" with 6 mentions (7%), Fjeld et al. (2020) explicitly undertook the task of analyzing CoCs to "map a consensus" (on the importance of principles) within the industry and the relevant governmental and NGO players. They, too, find principles similar to Jobin et al. (2019). They structure the content of the codes within "themes" which consist of values and principles that can reasonably be subsumed under aforementioned content. They list eight themes in total: privacy, accountability, safety and security, transparency and explainability, fairness and non-discrimination, human control of technology, professional responsibility, and promotion of human values. As mentioned above, a theme consists of a set of principles. In the case of "privacy," for instance, these principles are "Consent, Ability to Restrict Processing, Right to Erasure, [(Recommendation of)] Data Protection Laws, Control over the Use of Data, Right to Rectification, Privacy by Design, and Privacy (Other/General)" (ibid.). Hagendorff (2020) comes to similar results stating that "especially the aspects of accountability, privacy, or fairness appear all together in about 80% of all guidelines and seem to provide the minimum requirements for building and using an 'ethically sound' AI system" (ibid.). Floridi and Cowls (2019) identified five principles such as beneficence, non-maleficence, autonomy, justice, and explicability by analyzing six high-profile initiatives established in the interest of socially beneficial AI.

3.2 Codes of Conduct and Their Normative Features

Although there are overlaps in the listed values (Floridi & Cowls, 2019), such as privacy and fairness, the recommendations for action derived from these values vary: The normative concepts that are identifiable in the CoCs (henceforth: "values") differ in their accentuation of content depending on the originator (NGOs, GOs, companies, civil, and professional actors) (Zeng et al., 2018), on the addressed product (drones, social platforms, work tracking tools, …) as well as on the target group (technical companies, technical professionals, civil society, regulators, citizens). The tech-producer-user-differentiation highlights that each actor issues different CoCs targeting different interests and necessities arising from their products or users (Morley et al., 2019). This means that the respective CoCs always pertain to a certain perspective. Hence, analyzing and addressing ethical recommendations of actions need to take these distinctions into account: origin, product dependency, and target group. Due to this differentiation of interest and purpose, it is clear that striking differences exist in the prevalence of values as well as in their quality. Zeng et al. (2018) find that the average topic frequency differs depending on the nature of the actor (government vs. academia/NGO vs. corporations). The issue of privacy, for instance, is highly present in government-issued CoCs, but (statistically) significantly lower in CoCs issued in the academics and—even lower—in the corporate sector. These divergences can explain why CoCs converge on some core values, but at the same time differ tremendously in the emphasis they put on said values as well as on the respective subvalues. Hence, CoCs range from very abstract core values (such as justice or human dignity) to detailed definitions of technical approaches (e.g., data differentiation…) (see Jobin et al., 2019). Governmental CoCs, for example, support general and broad moral imperatives such as "AI software and hardware systems need to be human-centric" (EU Guidelines on ethics in artificial intelligence: Context and implementation (2018), p. 3) without further specification, whereas corporations tend to favor compliance issues when taking on privacy (Morley et al., 2019).

4 An Analytical Approach: Why Software Codes of Conduct Fail to Guide

The majority of CoCs agree on core values such as privacy, transparency, and accountability. Yet, CoCs diverge as soon as this level of abstraction must be supplemented with application-specific details or precise definitions of concepts. Moreover, we also encounter significant differences in the prioritization of values and a derivation of focal points. Given these observations, the question is then: If CoCs are in broad agreement on core values, why do they differ in their statements? One reasonable explanation is that the difference is rooted in the very nature of values, namely their underdetermination. This underdetermination is directly linked to the problem that CoCs are barely able to provide normative orientation in software development. This lack of orientation, in turn, triggers reactive behavior such as "cherry-picking," "risk of indifference," and "ex-post orientation," which we will discuss below. Combined, these issues result in a desire to rely on gut feeling, so-called heuristics, which seemingly support (ethical) decision making without much effort. Unfortunately, those shortcuts cannot provide a substitute for proper ethical deliberation and, thus, do not allow for a well-considered decision.

4.1 The Problem of Underdetermination

Numerous CoCs contain values that are central to the ethical handling of software and that can hardly be reasonably disputed, such as the respect for human dignity or the claim that technology should be developed to serve mankind (humanistic perspective, a philosophical stance that puts emphasis on the value and (moral) agency of human beings (see Nida-Rümelin & Weidenfeld, 2018; Nida-Rümelin, 2020)). Although the normativity of these values is by no means to be questioned or relativized and these values can certainly claim normative validity, it should be obvious that a reduction of an entire value system to these central (meta-)norms is neither sufficiently determined in a theoretical sense nor does it lead to immediate useful practical implications. Moreover, it is hard or impossible to deductively derive other values from these central values. In fact, they rather take on the role of general statements, which on their own cannot provide concrete and, thus, practical guidance. A normative value system outlined within a CoC is, therefore, very often underdetermined insofar as it cannot give clear instructions on what ought to be done in any specific individual case. As a result, CoCs lack practical applicability, because they do not offer normative orientation for specific ethical challenges that occur on a regular basis—meaning they fail to achieve what they were initially created for. To make matters worse, due to the sheer number of different values proposed in the Codes, a fitting ethical value system to justify any possible action can be easily found, since no ranking of values can be presented with regard to the specific case at hand.

Many CoCs contain a variety of values, which are presented in a merely itemized fashion. Without sufficient concretization, reference, contextualization, and explanation, software engineers are left to themselves in juggling different values and compliance with each and every one of them. The nature of the values puts them inevitably in tension with each other when applied to the reality of software engineering (e.g., privacy vs. transparency or autonomy/freedom vs. safety). More often than not, the implementation of values ultimately requires a trade-off. Consider the example of transparency and privacy: Both values are mentioned in the majority of the codes, yet, it is generally infeasible to fully comply with both values simultaneously. Figure 2 shows a graphical representation of possible trade-offs between them. As long as the product is located in the upper right corner of the graph, it is possible to

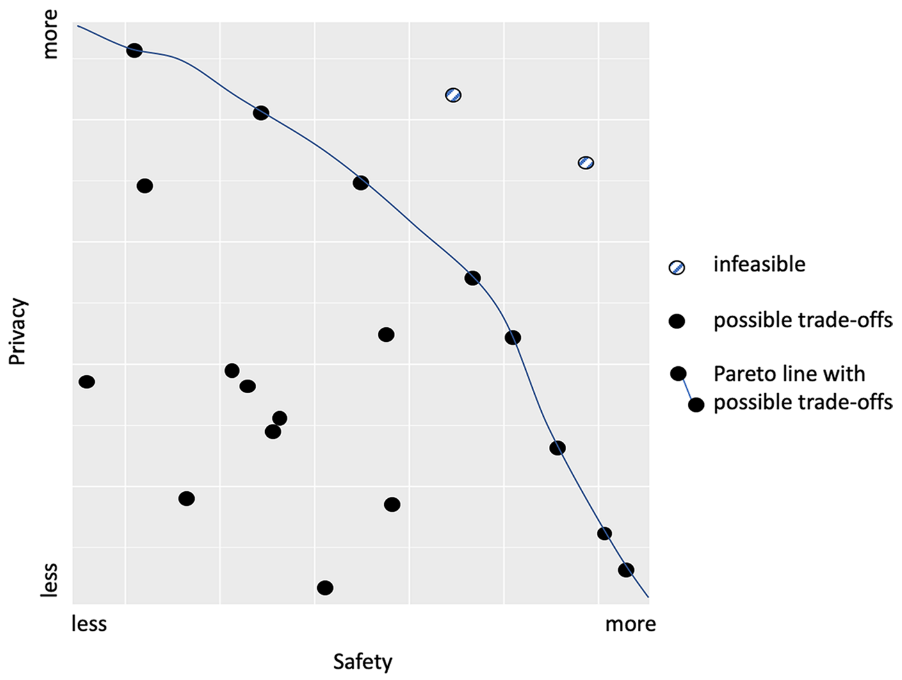

Fig. 2 Trade-offs between values and the efficiency frontier. Kearns and Roth (2019) make a similar point about trade-offs regarding accuracy and fairness in machine learning. In their case, the points would be machine learning models

improve the situation by either increasing compliance with one value or the other (or even both) which means moving closer towards the respective axis of the coordinate system (here: towards the origin). Once the line is reached, however, it becomes impossible to increase compliance with one value without decreasing compliance with the other. This line is known as Pareto optimality or the efficiency frontier. While it is certainly uncontroversial that the goal of software design should be a product that is efficiently optimizing the values we wanted to consider, it is by no means clear or obvious which point on the line, that is, which one of the many possible (Pareto optimal) trade-offs, should be implemented. Consider, for instance, the case of an app that enables geo-tracking. The trade-off between privacy and transparency looks completely different if this app is used to implement an anti-doping regime to monitor professional athletes or as a navigation app that is used by the average citizen to find the shortest route to her vacation destination. In the former, we might agree that professional athletes should give up more of their privacy in order to fight illegal doping, while we are appalled by the fact that the regular user of a navigation app is constantly tracked and monitored.

Yet, this is exactly the point where ethical deliberation and moral decision making come into play. CoCs, thus, do not offer any help to answer this question. In fact, they remain quiet about the very reason software engineers might consult them in the first place. The joint CoC from ACM and IEEE, for instance, states that “[t]

he Code as a whole is concerned with how fundamental ethical principles apply to a computing professional's conduct. The Code is not an algorithm for solving ethical problems; rather it serves as a basis for ethical decision-making. When thinking through a particular issue, a computing professional may find that multiple principles should be taken into account and that different principles will have different relevance to the issue. Questions related to these kinds of issues can best be answered by thoughtful consideration of the fundamental ethical principles" (Gotterbarn et al., 2018). As long as we deal with win–win situations, CoCs can be applied, but are of little use. As soon as we reach the problem of weighing legitimate ethical reasons and values, they become rather useless. It is, therefore, unclear what it means that CoCs serve as the basis for ethical decision making when in fact the normative deliberation of the software engineer herself would constitute the footing of ethical behavior.

4.2 Unwanted Behavior as a Result of Underdetermination

Cherry-picking Ethics Once Pareto optimality is achieved, any increase of compliance with one value must result in a decrease of compliance with the opposing one. It follows that in the applied case many different actions can be justified with recourse to various values from the same CoC (e.g., individual privacy vs. societal welfare). The CoC then becomes a one-stop shop offering an array of ethical values to choose from depending on which principle or value is (arbitrarily) deemed relevant in a certain situation (Floridi, 2019). Coherent ethics, however, require that ethical theory needs to cohere *externally* with our moral and general experience, beliefs, and conventions. Only then can ethical theory give an account of diverse daily (normative) experiences while preserving *internal* coherence within an ethical theory, e.g., using the utilitarian principle to form a consistent system of interrelated parts (De George, 2013). Coherent ethics cannot be realized if codes of conduct are understood as arbitrary accumulations of values, from which we can select values more or less at random. Consequently, CoCs are unhelpful for solving difficult decision situations as they almost always offer the easy way out: there will always be a value which is easy to identify or cheap to apply and implement. That is why CoCs lack normative guidance. This is also supported by basic economic theory and experience. People usually choose the path of least resistance or the cheapest implementation (Judy, 2009). Unfortunately, this arbitrariness and thriftiness make it virtually impossible to achieve a well-founded, coherent normative perspective.

Risk of Indifference Many CoCs are often underdetermined and offer the possibility that any one particular Code of Conduct could be used to justify different and even contradictory actions. Thus, many Codes of Conduct could foster the danger of ethical indifference (Lillehammer, 2017): They offer neither concrete nor abstract guidance and the normative function of the CoCs is anything but guaranteed. Additionally, most codes state obvious and uncontroversial values and ethical goals. In fact, their generic nature leaves the reader with the feeling that their gut feeling and practical constraints should have the final verdict when it comes to trade-offs.

Ex-post Orientation In addition to the problem of broadly stated general values lacking practical orientation (meta-values), it is important to understand ethics not as a restriction of action, but as an orientation or as an objective towards which the action should be directed so that shared ways of life can be supported (Hausman, 2011). However, since CoCs provide values that need to be considered, but which are underdetermined due to their normative character, desired ethical values in their abstraction have little influence on the development process. The reason for this is that values are not process-oriented and do not include logically the means by which they can be achieved. This very nature of values may lead to the fact that values are often considered only afterwards and just adapted to actions, but do not align action accordingly. This is especially true for the domain of software engineering in which every newly developed tool is very context specific and it might be harder to bridge from abstract principles to the concrete situation compared to other forms of engineering. It is important to stress that ethical deliberation is more than weighing conflicting values and assessing consequences. We need to think about the desirability of objectives as well as the normative orientation of action contexts into which technical artifacts are integrated and we must consider normativity in the course of system development. In the course of conceptualizing technical feasibility, we must address ethics from within since "(e)thics on the laboratory floor is predicated on the assumption that ethical reflection during research and development can help to reduce the eventual societal costs of the technologies under construction and to increase their benefits" (van der Burg & Swierstra, 2013, p. 2). This is precisely what we hope to achieve with our EDAP scheme.

The Desire for Gut Feelings The underlying motivation or the desired goal of a CoC or an ethical guideline in general could be to serve as a heuristic. Heuristics simplify and shorten the deliberation process in order to facilitate decisions. Gigerenzer and Gaissmaier (2011) describe them as "efficient cognitive processes, conscious, or unconscious, that ignore part of the information. [...] using heuristics saves effort." Yet, the use of heuristics in the moral domain seems to be distinctive to their application elsewhere, since they are based on "frequent foundation of moral judgments in the emotions, beliefs, and response tendencies that define indignation" (Sunstein, 2009). One prominent example is the connection between disgust and moral judgement (Landy & Goodwin, 2015; Pizarro et al., 2011). Especially, if novel situations and previously unseen problems arise, there is little reason to believe that a heuristic that might have been a good fit in previous cases will also fit well into the new context. As soon as uncertainties arise because the objectives of actions are conflicting and no unerring automatic solution can be achieved by applying dispositions, conventions, or moral rules, the resulting lack of normative orientation must be resolved by reflection (Dewey, 1922; Mead, 1923).

In sum, the underdetermination of values due to their universal character makes it impossible to deduce all possible specific, concrete applications of said value. Therefore, software engineers may make a rather arbitrary and impromptu choice when it comes to the values they want to comply with: picking whatever value is around or—as economists would say—in the engineer's relevant set and which often justify actions that they want to believe to be right (this effect is called motivated reasoning, see Kunda (1990) and Lodge and Taber (2013)). And because of these two aspects—the lack of specificity as well as the resulting cherry-picking-mentality—we encounter an attitude of indifference (Spiekermann, 2015). Furthermore, in many cases, the system is only checked for normative issues at the end of the development process as part of the technology assessment (ex-post orientation). This tendency will most likely not lead to a change in preferences and the conceptualization of a software system is inconsistent in terms of its normative dimension as this process disregards normativity from the outset. Ultimately, this results in a desire for relying on gut feeling and not putting too much thought into it when deciding on vague ethical issues. Consequently, we encounter diffuse and unjustified normative statements that are barely reliable. However, if we want to shape our world responsibly, we must deliberate rationally to understand and justify what we are doing for which cause.

4.3 Codes of Conduct Cannot Replace Ethical Reflection

At this point, we want to sketch out an alternative to the reliance on abstract values in the form of CoCs and instead relocate ethical deliberation deep into the development process. As a first step to offer reasonable and well-founded ethical guidance, the values affirmed in the CoCs must be made explicit and be classified with regard to their respective context-dependent meaning as well as functional positions. Tangible conflicts must be resolved and formed into a coherent structure that guides action (Demarco, 1997). This process is a genuinely deliberative one, which means it cannot be reasonably expected to be successful by using heuristics or detailed specifications that can be provided ex ante. In fact, it is not possible to classify decision making rules with regard to individual cases, because values as such are context-independent. Software engineers may, thus, need assistance in their technical development—speaking philosophically to deliberate on issues rather casuistically than applying ethical principles (Jonsen & Toulmin, 1988). And this is exactly what we are aiming for by implementing a systematic approach to individual cases. For instance, the value "privacy" needs to be handled differently depending on whether the context is technical or political. While privacy issues might be addressed technically (e.g., there might be a possibility to store or process data while ensuring privacy), the political discourse about the question of what level or form of data collection itself is desirable remains unanswered. Thus, technological artifacts cannot be just evaluated from one perspective only, but need to be assessed against the backdrop of a multitude of categories such as authority, power relations, technical security, technical feasibility, and societal values (see also Winner, 1977). Hence, an "ethical toolbox" in such a simple form can hardly exist. The ethical deliberation process can neither be externalized nor completely delegated as the example of Google and their handling of the "right to be forgotten" requests nicely illustrates (Corfield, 2018). For at least some requests to delete a result from the search, it ultimately came down to a software engineer who flagged it as a bug and effectively made the final decision without the legal team conducting a final review. Therefore, delegation cannot be the single solution to ethical questions, because the ethical decision often falls back on the engineer for very practical reasons (specialists are expensive or they face an overwhelming workload; there might even be a shortage of ethicists who possess enough domain knowledge in software engineering). On the contrary, ethical reflection and deliberation can and must be learned and practiced. *It is a skill rather than a checklist* (see also Wedgwood, 2014). Thus, the political goal of developing both a general and a specifically effective CoC cannot be methodologically separated from a practice of ethical deliberation and reflection, at least not as long as one is not willing to give up on the notions of usability and impact. This requires, from an ethical perspective, to raise awareness among all involved—software developers in particular—for ethical deliberation and its integration into the very process of the production of software systems. Especially regarding software development, the task of embedding ethical deliberations to agile environments with an emphasis on team empowerment (like SCRUM) seems to be a reasonable, worthwhile, and necessary approach (see also López-Alcarria et al., 2019). Although external ethical expertise can be called upon to guide the common discourse, active ethical deliberation remains irreplaceable. This is also the conclusion of the High-Level Expert Group on Ethical AI:

> "Tensions may arise between [...] [ethical] principles, for which there is no fixed solution. In line with the EU fundamental commitment to democratic engagement, due process and open political participation, methods of accountable deliberation to deal with such tensions should be established" (EU High-Level Expert Group on Artificial Intelligence, p. 13).

While McLennan et al. (2020) suggest to include an ethicist in each project and every development team seems advantageous, it is hardly sensible or feasible. Not only would this produce unjustifiable costs for software companies but, with the increase in software development, a trend which will only grow in the future, there will also probably be an actual shortage of capable ethicists with a basic understanding of software development. In order to tackle normative questions adequately, it is, therefore, crucial to train software engineers in ethical issues and to implement a systematized deliberation process. This may very well be achieved by ethicists consulting on ethical deliberations in software development, yet the main goal remains: the empowerment of the individual development team regarding ethical deliberation and to implement ethics as a skill.

5 Ethical Deliberation Leads to Good Normative Design

Applied normative deliberation requires a structured, guided, and systematic approach to the assessment of values, their trade-offs, and their implementation (Zuber et al., 2020). Reflecting ethically upon technical artifacts to justify which features are reasonable is not a simple task, since technology is neither only a means to a given end, nor is it an end in itself. It also structures our social life. However, as Mulvenna et al. (2017) point out, "[w]hile most agree that ethics in design is crucial, there is little effective guidance that enables a broader approach to help guide and signpost people when developing or considering solutions, regardless of the area, market, their own expertise." This is also due to the fact that there is no such thing as *one* ethical approach that offers a strict principle or line of thought that is applicable to *all* normative questions with regard to information technological objects (e.g., this is one of the main problems with the question of fully autonomous driving: There is no single ethical principle that fully satisfies all normative positions and could be implemented). The field of digital ethics already covers many different questions, ranging from computer ethics discussing normative features of the professional ethos to machine ethics covering questions of how to design moral machines. Moreover, in applied ethics deontological or utilitarian principles are often used for a case evaluation. Yet, designing objects requires identifying moral issues and to react to said issues without limiting oneself to an in-detail argumentation that covers only a single ethical perspective. Take our daily routines and actions: we rarely evaluate all of our options only in terms of their consequences or if they fit some universality test as Kant suggested. Making virtues the sole basis of our actions is also insufficient as some cases require reflection of the effects of our actions (c.f. Nida-Rümelin, 2001, 2020; Ross & Ross, 1930). What is more, it is not possible to deduce all of its applications logically or analytically from a single (meta-)value. Therefore, the more software systems affect aspects of our daily lives, the greater the demand for thoughtful engineering practice in the form of a guided process to ensure ethically sound software. Instead of focusing on the search for a single ethical theory that is universally applicable, we need to introduce a practice of ethical engineering that is founded in theory. Expertise in ethical deliberation is pivotal for identifying potential ethical issues and addressing them properly throughout the software development process. However, this is not to be confused by an understanding of ethics as a theory or pure science, but rather as a way of dealing with normative matters.

This is what we intend to do in our framework "Ethical Deliberation in Agile Software Processes" (EDAP) (Zuber et al., 2020). This approach seeks to normatively align technical objects in a targeted manner, since it enables a goal-oriented, rational handling of values in technology design. Thus, we focus on the identification of values, their desirability, and, finally, their integration into software systems. We try to achieve this in three steps: (1) descriptive ethics, (2) normative ethics, and (3) applied ethics. Each step must be considered in relation to the concrete software application to be constructed.

(1) Descriptive ethics should facilitate access to the world of values: which values serve as orientation? Descriptive value analysis, which has emerged from the CoCs, is thus integrated into the deliberation process and serves primarily as guidance. Software developers become aware of the relevant topics within the industry in particular and society in general and identify their companies, societies, and their own values regarding the object in question.
(2) Normative ethics will scrutinize the selected values, evaluate them, and subject them to an ethical analysis: Are the software features under construction desirable in so far as we would like them to be applied in all (similar) situations (test of universality)? Do the benefits outweigh the costs given an uncertain environment (test of consequentialism)? Which attitudes do software developers, users,

or managers address explicitly and implicitly? Which dispositions should the designed software program promote? Which desirable attitudes are undermined (test of virtue ethics)? And finally, how do these answers fit into the desirable life, which means the optimization of our decision not only in regard to its consequences but also as the most choiceworthy action in regard to the way of life we favor (Rawls, (1971) 2009; DePaul, 1987; Gibbard, 1990; Wedgwood, 2017; Nida-Rümelin, 2019, 2020).

(3) Applied ethics, then, has to achieve even more: not only must one evaluate the individual case in order to reach a decision, but the final normative judgment must also account for technical possibilities and limitations—in other words, any solution must be technically implementable. The latter is the interface to value-sensitive design, which is an "approach to the design of technology that accounts for human values in a principled and comprehensive manner throughout the design process" (Friedman & Hendry, 2019; Friedman et al., 2002; Nissenbaum, 2005). It is, therefore, a form of technically implemented ethics and—at the same time—an imperative to the technician herself: Be value-sensitive! For this reason, we advocate the name "normative design."

It is of the utmost importance that if human–machine interaction allows for human values to be respected, technological artifacts must trace an image of the possible normative distortions or amplifications of attitudes, rules, or behaviors that result from them: It must be made clear how artifacts structure attitudes and actions. This calls for an ethical analysis. George Herbert Mead emphasizes that ethics can only suggest the method of dealing rationally with values and that these, in turn, are dependent on specific circumstances:

> The only rule that an ethics can present is that an individual should rationally deal with all the values that are found in a specific problem. That does not mean that one has to spread before him all the social values when he approaches a problem. The problem itself defines the values. It is a specific problem and there are certain interests that are definitely involved; the individual should take into account all of those interests and then make out a plan of action which will rationally deal with those interests. That is the only method that ethics can bring to the individual. (Mead, 1934).

To highlight the impossibility of (logical) reductiveness and, thus, the necessity of rational deliberation, consider again the following scenario from the beginning: A software and robotics engineering team is tasked to construct a nursing robot for the elderly. This system is supposed to take care of individuals, i.e., help them when they fall, contact the ambulance in case of an emergency, etc. The first step is to hermeneutically understand the scenario. In order to create a normatively appropriate system, it is important to locate the essential desirable values and discuss their exclusivity. To this end, one must identify normatively desirable "anchor points" and consequently examine them in regard to their relationality. Values and their interconnectedness cannot simply be logically derived from the description of the situation. In the nursing-robot case, we are concerned with the desirable objective of being able to lead a self-determined life for as long as possible. Regarding technical features, such a system may include a built-in camera and an audio microphone to receive voice commands to track the patient's position in order to guarantee an adequate reaction, e.g., calling for help in case of an emergency or assisting in getting back up if no injury is detected. Additionally, the stored data may give doctors the possibility to check for progression of dementia or other illnesses or to detect potential diseases through interpretation of these images using artificial intelligence. Much else is conceivable depending on objectives, funding, technological progress, and potential legal issues. Normatively speaking, the system enhances desired core values such as autonomy and wellbeing. A nursing robot that is not only technically robust and safe, but also constructed in accordance with the relevant normative standards still classifies critical situations, but this is done in a respectful, humane manner. In order to create systems that are—at least in principle—normatively adequate, we must ask which values are at risk when promoting autonomy by implementing a camera or audio-recording system. In this specific case, privacy concerns seem dominant. Thus, we have to ensure privacy while enabling autonomy which means, for instance, that people may not want to be filmed naked and do not want to disclose the location of their valuables, etc. Furthermore, people may want to know who has access to the data, what kind of data is stored, why, and where. In a perfect scenario, access to the data is exclusively limited to doctors who may use it to predict and treat illnesses (medical prevention). Even these basic normative considerations cannot be logically derived from the premise of promoting welfare and autonomy. We need to reason normatively to understand and highlight the ethical issues at hand while simultaneously using our empirical knowledge of the world. Consequently, technical solutions such as visual or audio-recording systems must be developed in such a way that they would not record certain scenes at all or recording, when medically necessary, must ensure a fair autonomy-privacy ratio by using specific techniques such as cartooning or blurring (Padilla-López et al., 2015). Other normative issues result from using certain techniques that may meet transparency requirements. Users want to know how recommendations are made and what reactions they can expect. A non-technical normative deliberation is whether it is desirable to live with an assistance system even if the system considers relevant normative aspects. These kinds of ethical concerns are not addressed within our ethical deliberation tool as we already outlined at the beginning. We focus on software development from the engineer's perspective. Whether or not there should be a nursing robot in the first place is a question that falls into the domain of (business) ethics and cannot be decided on the level of the software developer. As mentioned at the beginning, management needs to tackle these questions and consider the legal framework as well as other basic norms of society. The ethical deliberation of the developer, however, begins with the specific implementation of the technical object. For instance, the above-mentioned need for visual observation seems to be obvious—after all, the robot needs to "see" its surroundings in order to be a useful tool. Yet, there are many ways to implement the visual capabilities of the robot and to safeguard the privacy of individuals (cartooning, blurring, etc.). At this point, the software developer usually has the freedom (and, therefore, the responsibility) to deliberate on which specific implementation to use.

At this point, we would like to go beyond Van der Burg and Swierstra (2013) and highlight that not only societal costs may be reduced but profits of the developing firm may be increased. Taking normative issues seriously from the very beginning of a development project may lead to more effective systems and processes—especially in the long run. We must understand that technology transforms and systematizes our lives. And that it does so with a certain requirement for behavioral adaptation by the user or those affected by the system in order to ensure the functionality of the system. Hence, technology normatively structures the world we live in. This means we need to take normativity as we encounter it in our daily lives seriously and think about the technical artifact as being a part of it. Only then can we understand that an ethical deliberation is not only a deliberation of trade-offs. Moreover, we can highlight how the artifact can be normatively integrated into our daily routines. This means also to think about the compatibility of algorithms, data sources, and inputs, such as control commands, and also front-end design, to make sure that some desired normative features are technically well met (Friedman & Hendry, 2019; Simon, 2012). Normativity, thus, is not to be understood as some qualifier, but rather as a condition that enables specific desirable practices of daily life (Rip, 2013).

6 A Paradigm Shift: from CoCs to Ethical Deliberation in the Software Development Process

CoCs are difficult to use as normative guidelines for technical software development due to their underdetermined character. They may trigger behavior such as indifference or the cherry-picking of specific ethical values. Thus, they are of little immediate use during the software development processes. Since CoCs lack direct real-world applicability, ethical values may only be chosen after the product is finished depending on which values "fit" (are most compatible with the existing product). However, in order to build an ethically sound system, it is essential to consider normative issues during the development process. Only when integrating values from within the development process, engineers will be able to consciously build software systems that reflect normative values. This will foster an understanding of how technological artifacts act as normatively structuring agents and improve the chances of creating ethically informed software systems. In contrast to related work, we suggest moving from a simple application of "ethical heuristics" to a point where we treat ethical thinking as a skill that has to be practiced and can be embedded deeply into the software development process. Consequently, ethical deliberation must not be limited to ethics councils, company advisory boards, or other special committees. Rather, it needs to be practiced and shaped by the software developers who create and intricately understand the technical system. This approach would lead to ethical empowerment of software engineers as part of the general empowerment trend that we see in agile software development. It is important to outline that developers should only be concerned with ethical issues that belong to their field of competence and the area they can actually affect. Ethical issues outside the scope of a single developer can then be delegated to other instances of the organization. In this sense, the ethical deliberation of software developers can also lead to a positive form of whistle blowing when an ethical issue is detected during the implementation process that has gone unnoticed by decision makers of the company.

In order to successfully implement ethical deliberation into the software engineering practice, we need to answer two main questions: First, why should companies build “ethical software” and second, how should they do it?

Since ethical deliberation requires a willingness to invest time and resources, a company has to encourage and support its engineers to consider ethical issues and discuss different ways to develop a product. Further research is necessary to understand the incentives that might motivate or hinder a company to implement processes that will help to address these concerns.

Proper ethical deliberation might, for instance, initially decelerate time to market a product. However, once a product has been built based on an ethical deliberation, the inclusion of relevant stakeholders might prove to be a key factor regarding customer loyalty and appreciation. This is in the spirit of agile software development where it was found that “most defects end up costing more than it would have cost to prevent them” (Beck & Andres, 2004, Ch. 13). Similarly, ethically informed products might increase a company’s reputation, promote its success, and might ultimately even help in creating better products. Unfortunately, it is unclear where to best embed such an ethical deliberation into the development process. Agile processes like Scrum suggest holding meetings at the beginning and the end of a sprint and it seems likely that these meetings offer a natural venue to discuss ethical issues. This, however, has so far not been empirically validated. The next move, therefore, is to offer an approach that provides guidance on how to implement an ethical deliberation into the concrete processes of agile development and to empirically research the conditions under which ethically informed software engineering can reach its full potential.

Funding Open Access funding enabled and organized by Projekt DEAL. This study was financially supported by the Bavarian Research Institute for Digital Transformation. Declaration statements are not applicable.

References

Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change*. (2nd ed.). Addison-Wesley Professional.

Boenink, M., Swierstra, T., & Stemerding, D. (2010). Anticipating the interaction between technology and morality: A scenario study of experimenting with humans in bionanotechnology. *Studies in ethics, law, and technology*, *4*(2).

Brey, P. A. (2011). Anticipatory technology ethics for emerging IT. *CEPE 2011: Crossing Boundaries*, *13*, 868.

Brey, P. A. (2012). Anticipatory ethics for emerging technologies. *NanoEthics, 6*(1), 1–13.

Corfield, G. (2018). *Here is how Google handles Right To Be Forgotten requests*. The Register. Link: https://www.theregister.com/2018/03/19/google_right_to_be_forgotten_request_process/#:~:text=RTBF%20trial%20Google%20allows%20software,its%20internal%20bug%2Dhandling%20systems.

Davis, M. (1998). Thinking like an engineer. *Studies in the Ethics of a Profession*, 143.

De George, R. T. (2013). Ethics and coherence. *The American Philosophical Association Centennial Series*, 717–732.

DeMarco, J. P. (1997). Coherence and applied ethics. *Journal of Applied Philosophy, 14*(3), 289–300.

DePaul, M. R. (1987). Two conceptions of coherence methods in ethics. *Mind, 96*(384), 463–481.

Dewey, J. (1922). *Human nature and conduct*. Henry Holt and Company.

Digital.ai. (2020). *The 14th annual State of Agile survey (No. 14), Annual State of Agile Report. Digital.ai*.

Drury, M., Conboy, K., & Power, K. (2012). Obstacles to decision making in Agile software development teams. *Journal of Systems and Software, 85*(6), 1239–1254.

EU High-level expert group on artificial intelligence (2018). Ethics Guidelines for Trustworthy AI. https://ec.europa.eu/futurium/en/ai-alliance-consultation/guidelines#Top. Accessed 20 Apr 2021.

Eubanks, V. (2018). *Automating inequality - How high tech tools profile, police, and punish the poor*. St. Martin's Press.

Fjeld, J., Achten, N., Hilligoss, H., Nagy, A., & Srikumar, M. (2020). Principled artificial intelligence: Mapping consensus in ethical and rights-based approaches to principles for AI. *Berkman Klein Center Research Publication*.

Floridi, L. (1999). Information ethics: On the philosophical foundation of computer ethics. *Ethics and Information Technology, 1*(1), 33–52.

Floridi, L. (2013). *The ethics of information*. Oxford University Press.

Floridi, L. (2019). *The logic of information: A theory of philosophy as conceptual design*. (1st ed.). Oxford University Press.

Floridi, L., & Cowls, J. (2019). A unified framework of five principles for AI in society. *Harvard Data Science Review*, 1(1). sie zeigen prinzipien auf, die in den meisten CoCs vorkommen, aber immer auf AI.

Friedman, B., & Hendry, D. G. (2019). *Value sensitive design - Shaping technology with moral imagination*. MIT University Press.

Friedman, B., Kahn, P., & Borning, A. (2002). Value sensitive design: Theory and methods. *University of Washington technical report*, (2–12).

Gibbard, A. (1990). *Wise choices, apt feelings: A theory of normative judgment*. Harvard University Press.

Gigerenzer, G., & Gaissmaier, W. (2011). Heuristic Decision making. *Annual Review of Psychology, 62*(1), 451–482.

Gotterbarn, D. W., Brinkman, B., Flick, C., Kirkpatrick, M. S., Miller, K., Vazansky, K., & Wolf, M. J. (2018). *ACM code of ethics and professional conduct*.

Hagendorff, T. (2020). The ethics of AI ethics: An evaluation of guidelines. *Minds and Machines*, 1–22.

Hausman, D. (2011). *Preference, value, choice, and welfare*. Cambridge University Press.

Hewlett, P. (2017). Agile is the new normal: Adopting Agile project management. *Hewlett Packard Enterprise Development LP*.

Jobin, A., Ienca, M., & Vayena, E. (2019). The global landscape of AI ethics guidelines. *Nature Machine Intelligence, 1*(9), 389–399.

Jonsen, A. R., & Toulmin, S. (1988). *The abuse of casuistry: A history of moral reasoning*. Univ of California Press.

Judy, K. H. (2009). Agile principles and ethical conduct. In *2009 42nd Hawaii International Conference on System Sciences* (pp. 1–8). IEEE.

Kaptein, M., & Schwartz, M. S. (2008). The effectiveness of business codes: A critical examination of existing studies and the development of an integrated research model. *Journal of Business Ethics, 77*(2), 111–127.

Kearns, M., & Roth, A. (2019). *The ethical algorithm: The science of socially aware algorithm design*. Oxford University Press.

Koning, T. & Koot, W. (2019). *Agile transformation: KPMG Survey on Agility*. KPMG. Retrieved from https://assets.kpmg/content/dam/kpmg/nl/pdf/2019/advisory/agile-transformation.pdf. Accessed 20 Apr 2021.

Kunda, Z. (1990). The case for motivated reasoning. *Psychological Bulletin, 108*(3), 480–498. https://doi.org/10.1037/0033-2909.108.3.480.PMID2270237.

Landy, J. F., & Goodwin, G. P. (2015). Does incidental disgust amplify moral judgment? A meta-analytic review of experimental evidence. *Perspectives on Psychological Science, 10*(4), 518–536.

Lillehammer, H. (2017). The nature and ethics of indifference. *The Journal of Ethics, 21*(1), 17–35.

Lodge, M., & Taber, C. (2013). *The Rationalizing Voter*. Cambridge University Press.

López-Alcarria, A., Olivares-Vicente, A., & Poza-Vilches, F. (2019). A systematic review of the use of agile methodologies in education to foster sustainability competencies. *Sustainability, 11*(10), 2915.

McLennan, S., Fiske, A., Celi, L. A., Müller, R., Harder, J., Ritt, K., & Buyx, A. (2020). An embedded ethics approach for AI development. *Nature Machine Intelligence, 2*(9), 488–490.

McNamara, A., Smith, J., & Murphy-Hill, E. (2018). Does ACM's code of ethics change ethical decision making in software development? In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 729–733).

Mead, G. H. (1923). Scientific method and the moral sciences. *International Journal of Ethics, 33*(3), 229–247.

Mead, G. H. (1934). *Mind, self and society*. (Vol. 111). University of Chicago Press.

Moor, J. H. (1985). What is computer ethics? *Metaphilosophy, 16*(4), 266–275.

Morley, J., Floridi, L., Kinsey, L., & Elhalal, A. (2019). From what to how: an initial review of publicly available AI ethics tools, methods and research to translate principles into practices. *Science and Engineering Ethics*, 1–28.

Mulvenna, M., Boger, J., & Bond, R. (2017). Ethical by design: A manifesto. In *Proceedings of the European Conference on Cognitive Ergonomics 2017* (pp. 51–54).

Nida-Rümelin, J. (2001). *Ethische Essays*. Suhrkamp.

Nida-Rümelin, J. (2019). *Structural rationality and other essays on practical reason* (Vol. 52). Springer.

Nida-Rümelin, J. (2020). *Eine Theorie praktischer Vernunft. Walter de Gruyter GmbH & Co KG.*

Nida-Rümelin, J., & Weidenfeld, N. (2018). *Digitaler Humanismus: eine Ethik für das Zeitalter der künstlichen Intelligenz*. Piper.

Nissenbaum, H. (2005). Values in technical design. *Encyclopedia of science, technology, and ethics*, 66–70.

Noble, S. U. (2018). *Algorithms of oppression: How search engines reinforce racism*. Combined Academic Publ

O'Neil, C. (2017). *Weapons of math destruction: How Big Data increases inequality and threatens democracy*. Penguin.

Padilla-López, J. R., Chaaraoui, A. A., & Flórez-Revuelta, F. (2015). Visual privacy protection methods: A survey. *Expert Systems with Applications, 42*(9), 4177–4195.

Palm, E., & Hansson, S. O. (2006). The case for ethical technology assessment (eTA). *Technological forecasting and social change, 73*(5), 543–558.

Pizarro, D., Inbar, Y., & Helion, C. (2011). On disgust and moral judgment. *Emotion Review, 3*(3), 267–268.

Rawls, J. (2009). *A theory of justice*. Harvard University Press.

Reijers, W., & Coeckelbergh, M. (2020). *Narrative and Technology Ethics*. Palgrave Macmillan.

Rip, A. (2013). Pervasive normativity and emerging technologies. In *Ethics on the laboratory floor* (pp. 191–212). Palgrave Macmillan.

Ross, D., & Ross, W. D. (1930). *The right and the good*. Oxford University Press.

Schwartz, M. (2001). The nature of the relationship between corporate codes of ethics and behaviour. *Journal of Business Ethics, 32*(3), captain 247-262.

Secretariat T. B. (2003). *Values and ethics code for the public service*. Available at https://www.tbs-sct.gc.ca. Accessed 20 Apr 2021.

Simon, J. (2012). E-democracy and values in design. In *Proceedings of the XXV World Congress of IVR*.

Spiekermann, S. (2015). *Ethical IT innovation: A value-based system design approach*. CRC Press.

Sunstein, C. R. (2009). Some effects of moral indignation on law, *Vermont Law Review*. *Vermont Law School., 33*(3), 405–434.

Vallor, S. (2016). *Technology and the virtues: A philosophical guide to a future worth wanting*. Oxford University Press.

Van der Burg, S., & Swierstra, T. (Eds.). (2013). *Ethics on the laboratory floor*. Springer.

Verbeek, P. P. (2005). Artifacts and attachment: A post-script philosophy of mediation. *Inside the politics of technology*, 125–146.

Verbeek, P. P. (2011). *Moralizing technology: Understanding and designing the morality of things*. University of Chicago Press.

Wedgwood, R. (2014). Rationality as a virtue. *Analytic Philosophy, 55*, 319–338.

Wedgwood, R. (2017). *The value of rationality*. Oxford University Press.

Whittlestone, J., Nyrup, R., Alexandrova, A., & Cave, S. (2019). The role and limits of principles in AI ethics: towards a focus on tensions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society* (pp. 195–200).

Winner, L. (1977). *Autonomous technology: Technics-out-of-control as a theme in political thought*.

Zeng, Y., Lu, E., & Huangfu, C. (2018). Linking artificial intelligence principles. *arXiv preprint*. https://arxiv.org/abs/1812.04814. Accessed 20 Apr 2021.

Zuber, N., Kacianka, S., Nida-Rümelin, J. & Pretschner, A. (2020): Ethical deliberation for Agile software processes: EDAP manual. Hengstschläger, M (ed.): *Digital Transformation and Ethics*.