Informatik Spektrum (2021) 44:348–354
https://doi.org/10.1007/s00287-021-01390-8

HAUPTBEITRAG

Ethik in der agilen Software-Entwicklung

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

Angenommen: 16. Juli 2021 / Online publiziert: 30. August 2021

Zusammenfassung
Nicht erst seit der anstehenden EU-Gesetzgebung zur Regulation von Künstlicher Intelligenz (KI) spielt die Berücksichtigung und Verwirklichung von Normen und Werten in technischen Systemen eine wesentliche Rolle. In diesem Aufsatz zeigen wir, dass, warum und wie agile Entwicklungsmethoden wie Scrum hervorragend geeignet sind, Werte in die Softwareentwicklung und in Softwareprodukte einzubetten; wie ein Entwickler so seine individuelle Verantwortung wahrnehmen kann; und wie die Diskussion um Ethik in der KI, auf Software allgemein erweitert, letztlich einen nächsten Schritt auf dem europäischen Weg zur digitalen Souveränität bedeuten kann.

Software, nicht KI!

Das starke gesellschaftliche Echo auf die Versprechungen der künstlichen Intelligenz und vor allem des Maschinenlernens hat zu einer erneuten Debatte zu Ethik in der Technik geführt, wie wir sie in dieser Breite zuletzt zur Präimplantationsdiagnostik, zum Klonen oder zur Nuklear- und Gentechnik erlebt haben. Die Kernfrage in solchen Debatten ist dann, welche Werte wir in der technischen Entwicklung und Forschung sowie in entsprechenden Produkten berücksichtigen können; und auch, ob und wie weit wir die Technologie überhaupt zulassen sollen. Das Grundproblem ist nicht neu, denn die Entwicklung von Technik wirft bereits bei Aristoteles philosophisch relevante Fragestellungen auf, die er in seiner *Nikomachischen Ethik* thematisiert [1]. Eine eigenständig systematische Herangehensweise der Technikphilosophie im modernen Sinne wurde 1877 durch Ernst Kapp eingeführt [11]. Seither firmieren technikphilosophische Betrachtungen und Ansätze unter verschiedenen Namen mit unterschiedlichen Facetten: Technikfolgenabschätzung, value-sensitive Design, responsibility-driven Design usw. [5, 8, 23]. Vor allem die Erkenntnis, dass Software zunehmend in allen Lebensbereichen Entscheidungen trifft oder zumindest den Systembenutzern Entscheidungsgrundlagen liefert, gepaart mit einem damit einhergehenden Gefühl des Kontrollverlusts, trägt zur aktuellen Popularität dieser neu aufgelegten Ethikdebatte bei.

Den Softwareingenieur muss es erstaunen, dass sich die aktuelle Ethikdebatte auf KI fokussiert und, mit wenigen Ausnahmen wie etwa dem IEEE P7000/D3-Draft [10], nicht auf Software im Allgemeinen. Natürlich gibt es spezifische mit KI einhergehende ethische Herausforderungen. Letztlich erscheint es aber für die Umsetzung von Werten sekundär, ob ein Stück Software algorithmisch oder datengetrieben implementiert wird (auch wenn offensichtlich im zweiten Fall ein weiteres Artefakt operativ berücksichtigt werden muss). Ob ein komplexer Algorithmus in einem verteilten System wirklich so viel weniger „gefährlich“ oder transparenter ist als ein gelernter Entscheidungsbaum oder ein neuronales Netz, darüber kann man vielleicht geteilter Meinung sein. In diesem Aufsatz wollen wir uns der Frage widmen, wie man bereits bei der Softwareentwicklung allgemein ethische Werte implementieren kann. Es stellt sich heraus, dass die agile Entwicklung einen hervorragenden Ausgangspunkt darstellt [25].

Codes of Conduct

In den letzten 10 Jahren sind über 100 Codes of Conduct zur Entwicklung von Software entstanden, die von berufsständischen Vereinigungen, Firmen, NGOs und Wissenschaftlern entwickelt worden sind. Diese Codes benennen im Wesentlichen allgemein akzeptierte Werte wie Teilhabe, Transparenz, Fairness, Letztentscheidung beim Menschen usw.

✉ Alexander Pretschner
alexander.pretschner@tum.de

[1] Bayerisches Forschungsinstitut für Digitale Transformation, München, Deutschland

[2] Technische Universität München, München, Deutschland

[3] Ludwig-Maximilians-Universität München, München, Deutschland

[6]. Da sie nicht aus völkerrechtlich verbindlichen Menschenrechten abgeleitet sind, können sie keine allgemeine Verbindlichkeit beanspruchen. Zudem zeichnen sie sich durch einen hohen Abstraktionsgrad aus, der ihre konkreten Implikationen oft im Unklaren lässt. In der Praxis des Software Engineering bieten diese Codes daher oft nicht das erhoffte Maß an praktischer Orientierung. Für die Ingenieure liefern sie keine unmittelbare Handlungsanweisung und lassen sie entsprechend ratlos zurück.

Enttäuschung ist allerdings eine Funktion der Erwartung, und die Erwartungen der Praktiker sind wohl zu hoch gesteckt. Wahrscheinlich liegt die nicht unmittelbare Anwendbarkeit in der Natur der Sache und ist nicht vermeidbar: Software ist allgegenwärtig und kontextspezifisch, und kontextspezifisch ist auch das Software Engineering [4]. Es erscheint fast absurd, dann zu erwarten, in einem Code of Conduct ein auf alle Kontexte passendes Werkzeug für die ethische Implementierung von Werten in Software zu finden. Das ist schlicht unmöglich. Deswegen muss die Einbettung von Werten in die Software(entwicklung) immer fallspezifisch geschehen.

Spielarten der Ethik

Die Ethik ist ein ungemein facettenreiches Feld der Philosophie [14]. In der Interaktion mit Ingenieuren und Studierenden der Informatik spielt zumeist implizit die sogenannte konsequentialistische Ethik eine gewichtige Rolle. Diese Theorie bewertet Handlungen hinsichtlich ihrer Konsequenzen, die sich aus den zu treffenden Entscheidungen ergeben werden. In der durch das Buch von Ferdinand von Schirach popularisierten Frage beispielsweise, ob ein von Terroristen gekapertes Flugzeug beim Anflug auf ein vollbesetztes Stadion abgeschossen werden darf, stellt die Antwort „lieber wenige Tote als viele Tote“ einen klar konsequentialistischen Standpunkt dar. Möglicherweise ist dieses bei Technikern oft intuitive Verständnis von Ethik ihrer Ausbildung geschuldet [21]: Sie werden trainiert, Probleme zu lösen und über Trade-offs und Konsequenzen nachzudenken.

Demgegenüber stehen beispielsweise deontologische Theorien, die wir hier vereinfachend zusammenfassen wollen. Diese Theorien bewerten Entscheidungen nicht bzgl. sich ergebender Konsequenzen als richtig oder falsch. Stattdessen erfolgt das Werturteil allein darauf, ob das moralische Gebot erfüllt wird, z. B. der Kant’sche kategorische Imperativ. Gemäß deutscher Rechtsprechung ist es im Beispiel tatsächlich so, dass die genannte Entscheidung zum Abschuss des Flugzeugs nicht getroffen werden darf, weil Menschenleben nicht gegeneinander aufgerechnet werden dürfen. Eine weitere prominente Theorie findet sich in tugendethischen Ansätzen, die nach wünschenswerten Verhaltensweisen fragen, und somit den guten Charakter adressieren und wie dieser sich in Lebenspraxen entfalten kann [17, 22].

Wir stehen nun vor der Herausforderung, ethische Theorien für die Entwicklung oder Beurteilung von Software fruchtbar zu machen. Das ist nicht offensichtlich, wenn moralisch wünschenswerte Software intuitiv schwer zu identifizieren und zu prüfen ist. Ein Grund dafür ist, dass es unmöglich ist, entscheidende Kriterien zu definieren, die immer in gleicher Weise erfüllt werden sollen. Ethische Überlegungen lassen sich deshalb nicht anhand von Checklisten oder mithilfe vordefinierter Antworten abarbeiten [6], was das Argument der Kontextspezifizität von Software noch unterstreicht. Daher bleibt es unabdingbar, jedes neue Designprojekt von Anfang an, während des gesamten Entwicklungsprozesses, seines Einsatzes sowie seiner Wartung kontinuierlich zu evaluieren. Oder anders ausgedrückt: Wir müssen normativ abwägen, urteilen und praktisch argumentieren. Es gilt herauszufinden, welche allgemeinen Prinzipien oder Werte wir als erstrebenswert ausfindig machen und wie man diese in diesem speziellen Fall anwendet. Dies bleibt eine Aufgabe des technologischen Urteilsvermögens [17, 20].

Interessanterweise setzen die genannten Codes of Conduct einzelne Werte oft ohne Begründung fest, was das Unbehagen von Ingenieuren angesichts der mangelnden Konkretheit mit erklären könnte. Die Implementierung dieser Werte ist auch wohlfeil, solange dies keine Widersprüche, Kosten oder Mühen zeitigt: Was ist schon gegen „Transparenz“ zu sagen? Nichts – bis zu dem Zeitpunkt, bei dem Transparenz mit Privatheit kollidiert. Es spricht auch nichts gegen die Entscheidung, ein bestimmtes Produkt etwa im Kontext unbemannter Luftfahrzeuge nicht zu entwickeln – aber die Diskussion wird zu dem Zeitpunkt komplexer, sobald das mit ökonomischen Konsequenzen und der möglichen Entlassung von Mitarbeitern einhergeht. Allein die deskriptive Formulierung von Werten ist also ganz offenkundig nicht hinreichend.

Wir argumentieren im Folgenden, dass die agile Softwareentwicklung eine fallspezifische Berücksichtigung von Normen und Werten ermöglicht, ethische Deliberationen fördert und somit die Lücke schließen kann, die die Codes of Conduct offen lassen und offen lassen müssen.

Individuelle Verantwortung des Software-Ingenieurs

Zuvor müssen wir kurz überlegen, wer bei der Entwicklung und Verwendung softwareintensiver Systeme eigentlich Verantwortung trägt. Im Spektrum von Systemen als eigentlich Verantwortlichen bis hin zum einzelnen Individuum gibt es mindestens 4 Akteure, die Verantwortung übernehmen können und müssen [15, 16, 18, 19]: Die Ge-

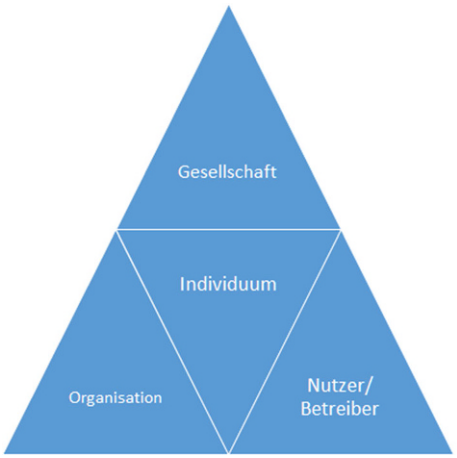

Abb. 1 Geteilte Verantwortung in soziotechnischen Systemen

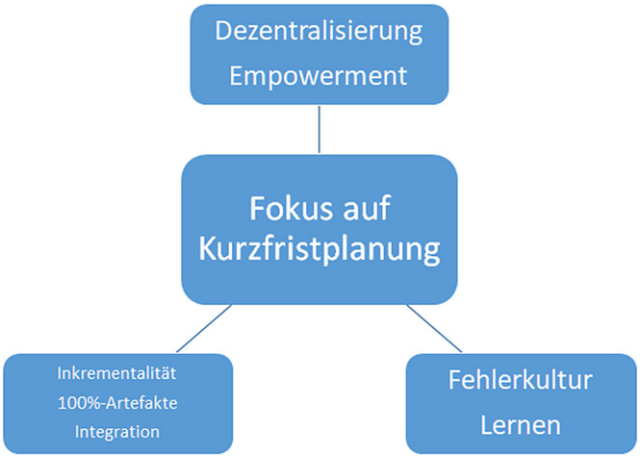

Abb. 2 Zentrale Phänomene der Agilität

sellschaft, die das System entwickelnde Organisation und ihre Teilbereiche, der einzelne Entwickler und der Betreiber sowie der Nutzer dieses Systems (Abb. 1). Eine spezifische Form der Gesichtserkennung etwa kann von der Gesellschaft akzeptiert oder verboten werden; eine Organisation kann sich entscheiden, Systeme zur Identifikation von Gesichtern anhand bestimmter Features zu entwickeln; ein Entwickler wählt Daten und Algorithmen aus; und der Betreiber oder Nutzer trägt für möglichen Missbrauch dieses Systems die Verantwortung. Pflegeroboter stellen ein weiteres klassisches Beispiel dar. Nur einige offensichtliche Beispiele für Softwaresysteme, die nicht datenzentriert und nicht primär auf künstlicher Intelligenz basieren, sind Kameraüberwachung ohne Gesichtserkennung, Kommunikationsdienste wie Telegram, Filesharing-Plattformen wie BitTorrent, digitale Währungen wie Bitcoin, elektronische Wahlen, barrierefreie Nutzerschnittstellen, die Auswahl von Präferenzen für Webseiten usw.

Softwareingenieure sind nun ganz offenkundig nicht allein verantwortlich. Und sie sind vor allem auch nicht für alle Externalitäten verantwortlich: Dass Airbnb zu Verwerfungen auf dem Wohnungsmarkt oder Uber zu einer Zunahme des nichtöffentlichen Verkehrs führt, dafür sind Software-Ingenieure nicht verantwortlich. Sie sind aber auch nicht *nicht*verantwortlich. Um die Wahrnehmung ihrer individuellen Verantwortung geht es in unserem Ansatz.

Agilität

Agile Softwareentwicklung und vor allem Agilität als Kultur lässt sich grob vereinfachend auf 4 wesentliche Phänomene reduzieren: Planung, Inkrementalität, Empowerment und Lernen (Abb. 2).

Erstens gibt es die Überlegung, dass bei der berühmten Konferenz zum Software Engineering 1968 in Garmisch zentrale Vorgehensweisen der Produktion von Industriegütern auf Software übertragen wurden. Ein Kern dabei sei die Trennung von *Planung* und Produktion, die sich dann in Ideen wie dem Wasserfall- und dem V-Modell niedergeschlagen haben. Software ist nun i. A. viel flexibler als Industriegüter und ermöglicht so die schnelle Reaktion auf sich ändernde Anforderungen und Kontexte. Unter anderem deswegen wurde diese Trennung in den 1990-Jahren durch Agilität prozessual rückgängig gemacht und die Planung mit der Produktion verwoben. Planbarkeit sei eine Illusion: „Developers and project managers often live (and are forced to) live a lie. They have to pretend that they can plan, predict and deliver, and then work the best way that they know to deliver the system“ [3]. Der Fokus wurde so von der Langfristplanung, die mit Artefakten wie Lastenheften und Pflichtenheften, Spezifikationen und Soll-Architekturen unterfüttert war, hin zu einer sehr präzisen Kurzfristplanung auf Sprint-Ebene verschoben, was mit einer Reduktion der zu entwickelnden Artefakte [2] einherging.

Die Erkenntnis, dass Langfristplanung in einer Welt schwierig ist, in der sich Anforderungen und Technologien ständig verändern (und wegen der Flexibilität von Software auch ändern *können*), führt zweitens fast zwangsläufig zu einer *inkrementellen Entwicklung*. Zentral ist hier die Idee, einzelne Funktionalitäten sequenziell vollständig zu Ende zu entwickeln und dann sofort mit dem jeweils bis jetzt entwickelten (ggf. Legacy-)System zu integrieren, was wie nebenbei die auftretenden kolossalen softwaretechnischen Probleme der Integration von Subsystemen erst nach längerer Zeit adressiert und gleichzeitig ein Schlüssel für Continuous Integration und Deployment ist.

Drittens verändert sich, auch als Konsequenz der Perspektive auf Kurzfristplanung, die Organisationskultur und das Verständnis über die Rolle von Mitarbeitern. In einer Weltsicht, in der feingranulare Spezifikationsdokumente den „Code Monkeys“ zur Bearbeitung übergeben werden,

gibt es Vorgaben treffende „höhere“ und ausführende „niedrigere“ Tätigkeiten. In einer agilen Welt, etwa Scrum, in der die wesentliche Vorgabe sich ändernde grobgranulare Anforderungen in Form sogenannter User Stories im Product Backlog und eben nicht Modulspezifikationen sind, wird dem Team von vornherein und gezwungenermaßen eine größere Gestaltungsfähigkeit zuteil, die sich nun auch auf unterschiedliche Facetten des Systems bezieht, was sich wiederum in crossfunktionalen Teams niederschlägt. Das Team ist im Vergleich zur Welt des Wasserfalls oder auch V-Modells ermächtigt, neudeutsch *empowered*, und verfügt über viel größere Gestaltungsfreiräume. Das Team entscheidet nicht nur, welche Features in einem Sprint bearbeitet werden, sondern auch, wie ein Feature entwickelt wird – und kann so Einfluss auf ethische Konsequenzen nehmen! Das wiederum hat direkte Konsequenzen für die Struktur der Organisation, denn es stellt sich die Frage, was in einer solchen Welt die Rolle von „Managern“ auf verschiedenen Hierarchieebenen ist und erklärt auch, warum agile Softwareentwicklung in nichtagilen Unternehmensstrukturen häufig nicht so funktioniert, wie man sich das gewünscht hätte. Die Möglichkeit der Eigenverantwortung durch Empowerment ist in unseren Augen übrigens auch eine Pflicht zur Eigenverantwortung.

Viertens ist eine zentrale Idee hinter agilen Arbeitsformen eine Fehlerkultur und eine Kultur des *Lernens*, die sich wiederum direkt aus der Fokussierung auf Kurzfristplanung ergibt: Hier werden Fehler passieren, und die Entwicklung von Funktionalität kann sehr wohl zeigen, dass ein gewählter (technischer) Weg so nicht weiter verfolgt werden kann. Wenn das akzeptiert wird und in diesem Sinn Fehler als normal empfunden und gemacht werden dürfen, müssen natürlich Mechanismen zum Lernen aus diesen Fehlern etabliert werden. Das schlägt sich in Scrum etwa in Reviews und Retrospektiven nieder und zeitigt insgesamt die Notwendigkeit einer ständigen empirischen Prozesskontrolle.

Wir möchten hier nicht den Eindruck vermitteln, dass Agilität der eine Hammer für alle Nägel ist – den gibt es nicht. Neben anderen stellen die Größe von Projekten, Domänen mit regulierten Entwicklungsprozessen und Zertifizierungen, Organisation und Logistik der Produktion hardwarebasierter Systeme sowie die Fähigkeit und der nicht immer ausgeprägte Wunsch von Mitarbeitern, eigenverantwortlich zu arbeiten, natürliche und lange bekannte Stolpersteine dar. Uns geht es stattdessen darum, wie Ethik in die Entwicklung von Software eingebaut werden kann, und es zeigt sich, dass die 4 genannten Facetten agiler Entwicklung dies auf sehr natürliche Art und Weise ermöglichen. Die duale Perspektive, inwiefern Charakteristika moderner (agiler) Softwareproduktion als solche ethische Konsequenzen zeitigen, wird von Gürses und Van Hoboken [9] untersucht.

EDAP: Ethical Deliberation in Agile Processes

Erstens haben wir gesehen, dass in Codes of Conduct formulierte Werte einleuchten und unabhängig von Projekt und Konsequenz in einer deontologischen Ethik begründet sein können. Gleichzeitig reicht die deskriptive Formulierung von Werten allein aber für die ethische Entwicklung von Software nicht aus. Insbesondere haben wir dargelegt, dass hier projektspezifisch vorgegangen werden muss. Zweitens übt Software immer eine direkte Wirkung auf unsere Lebenswirklichkeit aus, die auch konsequentialistisch bewertet werden muss. Drittens haben wir erläutert, dass es in agilen Kulturen und insbesondere in der agilen Softwareentwicklung ermächtigte Teams in zunehmend flacheren Hierarchien gibt, die in kurzen Zyklen anhand nur grober Vorgaben eigenverantwortlich Funktionalitäten entwickeln. So können und müssen Software-Ingenieure einen direkten Einfluss auf die Berücksichtigung von Werten durch Technik nehmen. Dieses normative Vorgehen ist zu großen Teilen aber erst dann möglich, wenn konkrete Designentscheidungen anstehen, wenn also Software bereits entwickelt wird, und nicht vollständig vor der Entwicklung. Viertens sind die ständige Reflexion und das Lernen fast gezwungenermaßen Teil einer agilen Kultur, in die sich ethische Überlegungen nahtlos einbetten lassen.

Im EDAP-Schema (Ethical Deliberation in Agile Processes [24]) haben wir diese Puzzleteile zusammengefügt (Abb. 3). Kernidee ist, dass wir zunächst deskriptiv vorgehen und uns an gesellschaftlichen und organisationsinternen Wertevorgaben ausrichten, also von einem durch Gesellschaft und Organisation definierten Rahmen ausgehen. Das heißt beispielsweise, dass durch Entwickler die Entscheidung nicht mehr fundamental infrage gestellt wird, ob der Geschäftszweck „Assistenzroboter für Ältere“ als solcher überhaupt akzeptabel ist. Im Verhältnis zwischen Product Owner und Auftraggeber werden zweitens ggf. projektspezifisch innerhalb dieses Rahmens zentrale ethische Werte identifiziert, die Teil des Product Backlogs werden. Das kann auf Basis bestehender (heute häufig westlich geprägter) Codes of Conduct ebenso wie kultur- und kontextspezifisch mit anderen Hilfsmitteln und Methoden erfolgen. Dabei spielen auch mögliche Konsequenzen des Einsatzes des Softwaresystems eine Rolle, und so sehen wir hier deutlich die Relevanz sowohl der deontologischen als auch der konsequentialistischen Ethik. In Abb. 3 bezieht sich dies auf die Phasen 1 und 2 des EDAP-Schemas.

Innerhalb jedes einzelnen Sprints geht es drittens darum, diese Werte durch geeignete Mechanismen zu implementieren. Dazu müssen sich Entwickler in jedem Sprint weiterhin über ihre Wertvorstellungen insgesamt klar werden und insbesondere konsequentialistisch über die Folgen einer gewählten Methodik, eines gewählten Lösungsansatzes, einer gewählten Architektur, einer gewählten Implementie-

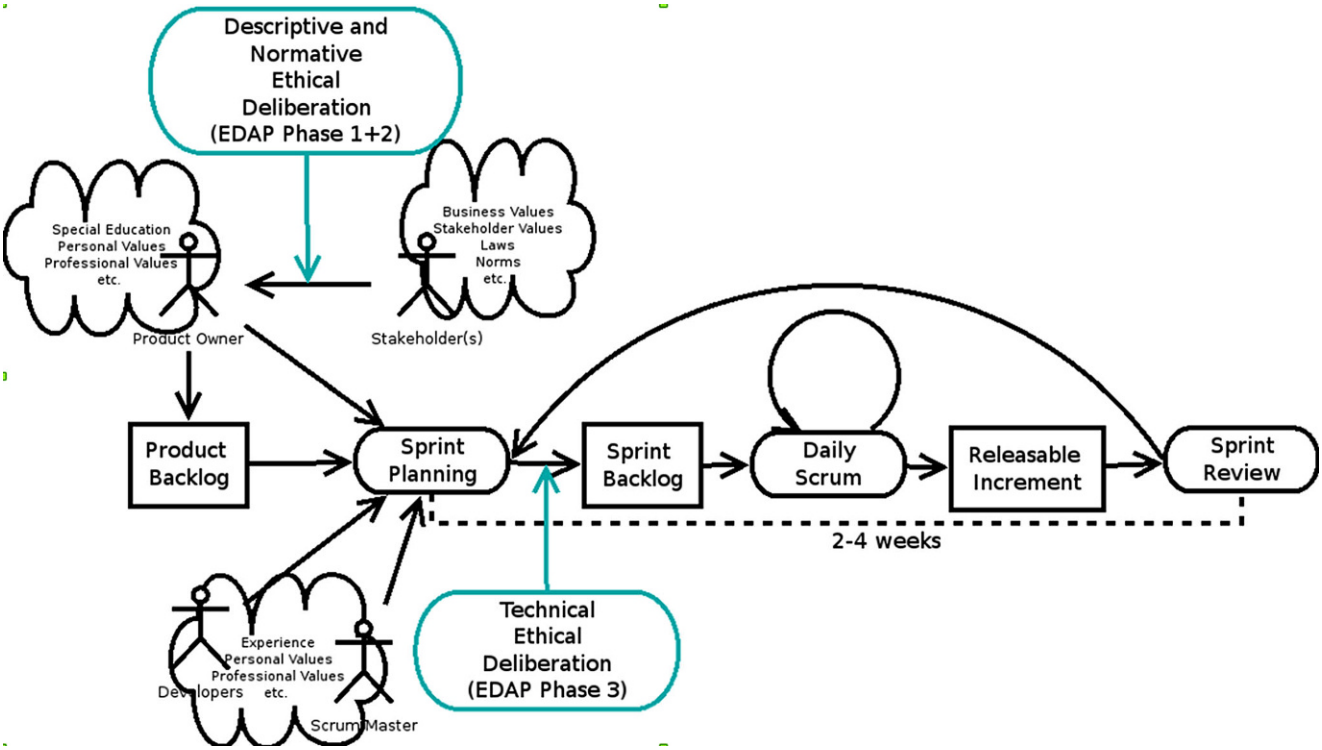

Abb. 3 Das EDAP-Schema und seine Einbettung in Scrum

rung oder eines gewählten Datensatzes nachdenken. Das geht an dieser Stelle viel konkreter als einmalig vor der Entwicklung, weil hier iterativ ein zunehmend ausführliches Verständnis des Systems entsteht. In der Abbildung ist dieser Zusammenhang mit der Phase 3 des EDAP-Schemas gekennzeichnet.

Wir haben oben gesehen, dass ethische Argumente interessant werden, wenn sie im Konflikt mit anderen ethischen oder ökonomischen oder sonstigen Argumenten stehen. Dies kann zu Dilemma-Situationen führen, die per Definition nicht aufgelöst, sondern nur entschieden werden können, weswegen es notwendig ist, diese Diskussion nicht beliebig zu verlängern, sondern eben innerhalb der Planung eines Sprints zu einem Ergebnis zu kommen.

Dieser Prozess ist ein kontextspezifischer Prozess des Reflektierens und der Deliberation, der entsprechend strukturiert und permanent durchgeführt werden muss (Abb. 4). Es gibt den Vorschlag, ständig einen „embedded ethicist“ [12] in Entwicklungsteams aufzunehmen. Das erscheint uns zu schwergewichtig, da dies für kleinere Unternehmen zu teuer wäre und auch generell ein Mangel an Menschen mit diesen Fähigkeiten besteht, der sich nicht schnell beheben lässt. Mit entsprechender Anleitung können die Entwicklungsteams diese Aufgabe bis zu einem gewissen Punkt selbst wahrnehmen. Genau darauf zielt das EDAP-Schema ab: Es umfasst die leichtgewichtige Identifikation, Lokalisation und kritische Reflexion relevanter Werte. Während erstere deskriptive Phasen darstellen, ist letztere bereits wesentlich normativ. Zugleich muss immer die technische Umsetzbarkeit oder eben absichtliche Abstinenz erörtert werden. Diesen Prozess können wir mit EDAP systematisch begleiten, um zu begründeten Entscheidungen und technischen Lösungen zu gelangen. Nicht immer bedarf es hierfür einer wissenschaftlich-ethischen Analyse, sondern oft gelangen wir auch mit vortheoretischem Wissen auf ein wünschenswertes Maß ethischer Deliberation.

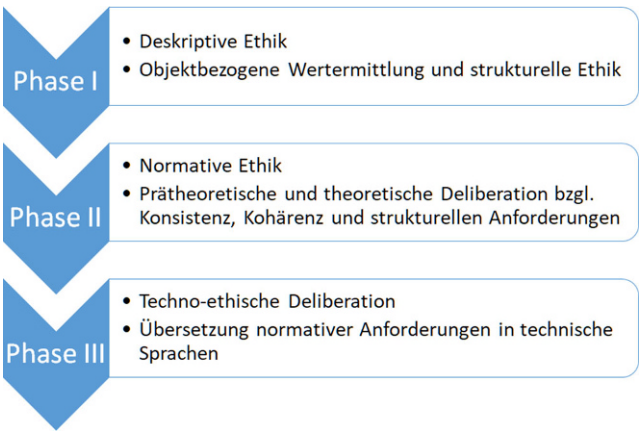

Abb. 4 Die Phasen des EDAP-Schemas

Einbettung in die Informatikausbildung

In den technischen Fächern gibt es zunehmend Bestrebungen, zentrale geistes- und sozialwissenschaftliche Zusammenhänge zu vermitteln, insbesondere im Zusammenhang mit Überlegungen zur Nachhaltigkeit. Es gibt verschiedene Ansätze mit widersprüchlichen Aussagen zu Wirksamkeit [13] und Definitionen der Vermittlung ethischer Zusammenhänge, die von Einführungsvorlesungen zur Ethik über gemeinsame Seminare von Philosophen und Informatikern hin zur tiefen und umfassenden Einbettung von Ethik in ein Informatikcurriculum [7] reichen. Wir wollen diese Diskussion hier nicht aufgreifen, aber kurz erwähnen, dass Programmierpraktika, in denen agile Entwicklung gelernt wird, sich offensichtlich sehr gut zur Einbettung ethischer Überlegungen eignen. An der Technischen Universität München haben wir mit solchen Kursen experimentiert und festgestellt, dass Techniker nach dieser Form von Horizonterweiterung geradezu lechzen; dass gleichzeitig das differenzierte Bewusstsein für ethische Zusammenhänge explizit geschaffen werden muss, was wie oben erwähnt vielleicht daran liegt, dass wir Studierende gerade zur Zielstrebigkeit beim Problemlösen erziehen; dass die Einbettung in einen technischen Programmierkurs anstelle einer einzelnen Vorlesung goutiert wird; dass Studenten zu Beginn ausnahmslos konsequentialistisch argumentieren; dass hier eine enge Betreuung durch Ethiker notwendig ist; und dass bzgl. Wirksamkeit und Attraktivität für die Studierenden insgesamt ein sehr positives erstes Resümee gezogen werden kann.

Zusammenfassung

Fassen wir die Zusammenhänge nochmals holzschnittartig zusammen. Agilität lässt sich durch die folgenden zentralen Ideen charakterisieren: (1) Langfristplanung kollidiert mit einer sich ständig ändernden Welt. (2) Kurzfristplanung führt schnell zu einer nichthierarchischen bzw. dezentralisierten Organisation. (3) Dezentralisierung führt zu Empowerment, Eigenverantwortung und großen Gestaltungsfreiräumen bei der Softwareentwicklung. (4) In solchen Strukturen sind Fehler wahrscheinlich und müssen durch explizite Mechanismen des Lernens adressiert werden. Aus ethischer Perspektive ergibt sich die folgende Argumentation: Aus der Eigenverantwortlichkeit der Entwickler entsteht (1) die Verpflichtung, auch ethische Zusammenhänge zu berücksichtigen, und zwar in Form von Werten und Konsequenzen, die sie durch Designentscheidungen beeinflussen können. Weil Entwickler eine knappe Ressource sind und ihren Arbeitgeber wechseln können, verfügen sie (2) über die Macht und die Möglichkeit, ethische Vorstellungen durchzusetzen. (3) In agilen Umgebungen dienen Werte als Grundlage einer inkrementellen Entwicklung und dann auch eines fortgesetzten Betriebs (DevOps), in der die Auswahl, Implementierung und ggf. kontinuierliches Monitoring von Lösungsansätzen, Daten und Algorithmen durch explizite Überlegungsprozesse fundiert und begleitet werden. (4) Agile Kultur ist aufgrund der sich aus der Kurzfristplanung ergebenden Fehler notwendigerweise reflektierend und lernend, und ein natürlicher Teil dieser Reflexionsprozesse können ethische Erwägungen sein. Im EDAP-Schema haben wir diese Deliberationsprozesse strukturiert, was dem einzelnen Ingenieur hilft, seiner individuellen Verantwortung gerecht zu werden.

Die Berücksichtigung ethischer Werte ist wie das Reflektieren über Nachhaltigkeit wohlfeil, sofern keine Kosten entstehen. Unser Ansatz erlaubt es, Entscheidungen zur Entwicklung von Systemen nicht binär mit „ja“ oder „nein“ zu beantworten und so „die Ethik“ aus der häufig als verhindernd wahrgenommenen Ecke herauszulösen. Ganz im Gegenteil räumen wir „der Ethik“ große gestalterische Möglichkeiten ein, indem verschiedene technische und methodische Mechanismen zur Implementierung dieser Werte erdacht und verwendet werden können. Gleichzeitig gibt es selbstverständlich ein Spannungsfeld mit der Innovationsfähigkeit, denn „Innovation muss schmutzig sein“ (Thomas Sattelberger), und je mehr Einschränkungen für den Entwurfsraum definiert werden, desto schwieriger wird Innovation offenbar. Wir sind davon überzeugt, dass sich hier Kompromisse finden lassen. Wir sind auch der Ansicht, dass die aktuell durch die EU stattfindende Regulierung von KI letztlich zu kurz greift, weil auch traditionelle algorithmische Systeme unseren Wertvorstellungen widersprechen können und dass gleichzeitig unser EDAP-Schema eine gute Grundlage für die derzeit diskutierte „Selbstzertifizierung“ für nichthochkritische KI-Anwendungen sein kann. Schließlich sind wir überzeugt, dass wir als Europäer selbstbewusst zu unseren Werten stehen und durch die Einbettung dieser Werte in Technologie als Alleinstellungsmerkmal durchaus im internationalen Wettbewerb der Systeme bestehen werden können.

Funding Open Access funding enabled and organized by Projekt DEAL.

Literatur

1. Aristoteles (2014) Die Nikomachische Ethik. De Gruyter, Berlin, Boston
2. Beck K, Grenning J, Martin R, Beedle M, Highsmith J, Mellor S, van Bennekum A, Hunt A, Schwaber K, Cockburn A, Jeffries R, Sutherland J, Cunningham W, Kern J, Thomas D, Fowler M, Marick B (2001) Manifesto for agile software development. http://agilemanifesto.org/. Zugegriffen: 7. Juli 2021
3. Beedle M, Devos M, Sharon Y, Schwaber K, Sutherland J (1998) SCRUM—an extension pattern language for hyperproductive software development. http://jeffsutherland.com/scrum/scrum_plop.pdf. Zugegriffen: 7. Juli 2021
4. Briand L, Bianculli D, Nejati S, Pastore F, Sabetzadeh M (2017) The case for context-driven software engineering research: generalizability is overrated. IEEE Softw 34(5):72–75
5. Friedman B, Hendry DG, Borning A (2017) A survey of value sensitive design methods. Found Trends Human Comput Interact 11(2):63–125
6. Gogoll J, Zuber N, Kacianka S, Greger T, Pretschner A, Nida-Rümelin J (2021) Ethics in the software development process: from codes of conduct to ethical deliberation. Philos Technol. https://doi.org/10.1007/s13347-021-00451-w
7. Grosz BJ, Grant DG, Vredenburgh K, Behrends J, Hu L, Simmons A, Waldo J (2019) Embedded EthiCS: integrating ethics across CS education. Commun ACM 62(8):54–61
8. Grunwald A (2010) Technikfolgenabschätzung: Eine Einführung Bd. 1. edition sigma, Berlin
9. Gürses S, Van Hoboken J (2017) Privacy after the agile turn. In: Polonetsky J, Tene O, Selinger E (Hrsg) Cambridge handbook of consumer privacy. Cambridge University Press, Cambridge
10. IEEE Computer Society (2020) P7000™/D3 draft standard for model process for addressing ethical concerns during system design
11. Kapp E (1877) Grundlinien einer Philosophie der Technik: zur Entstehungsgeschichte der Cultur aus neuen Gesichtspunkten
12. McLennan S, Fiske A, Celi LA, Müller R, Harder J, Ritt K, Haddadin S, Buyx A (2020) An embedded ethics approach for AI development. Nat Mach Intell 2:488–490
13. Mulhearn T, Steele L, Watts L, Medeiros K, Mumford M, Connelly S (2017) Review of instructional approaches in ethics education. Sci Eng Ethics 23:883–912
14. Nida-Rümelin J (2005) Angewandte Ethik: die Bereichsethiken und ihre theoretische Fundierung; ein Handbuch. Kröner, Stuttgart
15. Nida-Rümelin J (2011) Verantwortung. Reclam, Leipzig
16. Nida-Rümelin J (2017) Handlung, Technologie und Verantwortung. In: Berechenbarkeit der Welt? Springer VS, Wiesbaden, S 497–513
17. Nida-Rümelin J (2020) Eine Theorie praktischer Vernunft. De Gruyter, Berlin, Boston
18. Nissenbaum H (1994) Computing and accountability. Commun ACM 37(1):72–81
19. Nissenbaum H (1996) Accountability in a computerized society. Sci Eng Ethics 2(1):25–42
20. Rohbeck J (1993) Technologische Urteilskraft: Zu einer Ethik technischen Handelns. Suhrkamp, Berlin
21. Swierstra T, Rip A (2007) Nano-ethics as NEST-ethics: patterns of moral argumentation about new and emerging science and technology. Nanoethics 1(1):3–20
22. Vallor S (2016) Technology and the virtues: a philosophical guide to a future worth wanting. Oxford University Press, New York
23. Van den Hoeven J, Vermaas PE, Van de Poel I (2015) Handbook of ethics, values and technological design
24. Zuber N, Kacianka S, Pretschner A, Nida-Rümelin J (2020) Ethische Deliberation für agile Softwareprozesse. In: Rat für Forschung und Technologieentwicklung (Hrsg) Digitaler Wandel und Ethik. Ecowin, Elsbethen, S 150–176
25. Zuber N, Kacianka S, Gogoll J, Pretschner A, Nida-Rümelin J (2021) Empowered and embedded: ethics and agile processes. arXiv:2107.07249 (Submitted. Preprint available)