Mario Donick

Nutzerverhalten verstehen – Softwarenutzen optimieren

Kommunikationsanalyse bei der Softwareentwicklung

MOREMEDIA

Springer Vieweg

Nutzerverhalten verstehen – Softwarenutzen optimieren

Mario Donick

Nutzerverhalten verstehen – Softwarenutzen optimieren

Kommunikationsanalyse bei der Softwareentwicklung

Springer Vieweg

Mario Donick
Magdeburg, Deutschland

ISBN 978-3-658-28962-1 ISBN 978-3-658-28963-8 (eBook)
https://doi.org/10.1007/978-3-658-28963-8

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Vorwort: Read The Fucking Manual?

Liebe Leser*innen,

herzlich willkommen zu „Nutzerverhalten verstehen – Softwarenutzen optimieren: Kommunikationsanalyse bei der Softwareentwicklung“. Was erwartet Sie in diesem Buch?

Im Wesentlichen lernen Sie zwei Dinge:

(1) In den Kap. 1 und 2 entwickeln Sie ein Bewusstsein dafür, dass Software ein kommunikativer Ausdruck ist, der von Nutzer*innen in konkreten Situationen verstanden werden muss: Als Entwickler*in entwickeln Sie Produkte, die für Ihre Nutzer*innen mehr oder weniger relevant erscheinen können. Dass Softwareentwicklung damit auch ein Kommunikationsproblem zwischen Entwickler*innen und Nutzer*innen darstellt, wird bisher nur selten behandelt.
(2) In den Kap. 3, 4, 5 und 6 erlernen Sie ein Beobachtungsverfahren, mit dem Sie Nutzungssituationen besser verstehen können. So ein Verständnis ist wichtige Grundlage, damit Sie Software sensibler für Nutzer*innen und Nutzungssituationen gestalten können, also das oben benannte Kommunikationsproblem erfolgreich lösen können. Das Verfahren stammt aus der Kommunikationswissenschaft.

Dieses Buch ist damit etwas anders als andere Fachbücher zur Softwareentwicklung. Es liegt sozusagen quer zu den üblichen Entwicklungsmodellen der Informatik, und Sie können von dem Buch profitieren, egal ob Sie moderne agile Methoden verwenden, dem klassischen Wasserfallmodell anhängen oder nach dem Motto „Code & Fix“ (codieren und Fehler beheben) befahren. Viele Bücher, Verfahren und Modelle gehen davon aus, dass menschliche Kommunikation etwas ist, das irgendwie nebenher läuft, das wir schon irgendwie können oder das man schon irgendwie lernen kann, sofern man sich nur auf dieselben Begriffe einigt. Vielleicht macht man – um die Zusammenarbeit im Team zu verbessern – noch einen Kurs zu den Top 10 der Kommunikationspsychologie, wie etwa „Man kann nicht nicht kommunzieren“, dem „Vier-Ohren-Modell“ oder „gewaltfreier Kommunikation“. Das ist alles nützlich und wichtig. Aber leider nicht genug.

Erfolgreicher Zusammenarbeit, erfolgreicher Entwicklung und erfolgreichen Softwareprodukten liegt erfolgreiche Kommunikation zugrunde, aber erfolgreiche Kommunikation

ist mehr, als einen gemeinsamen Code zu teilen oder einige erlernte Tipps mechanisch zu beherzigen. Erfolgreiche Kommunikation verlangt zuallererst, deren Unwahrscheinlichkeit (so der Soziologie Niklas Luhmann) anzuerkennen: Es ist nicht selbstverständlich, dass Menschen einander verstehen, und dass Nutzer*innen Software verstehen.

Menschen untereinander sind wenigstens in der Lage, drohende Missverständnisse spontan zu bearbeiten und von Ausdruck zu Ausdruck Verständigung zu sichern. Aber im Umgang mit Software ist dieser elegante Weg versperrt. Anders als soziale Interaktion, die viele sogenannte Reparaturverfahren kennt, um den Erfolg von Kommunikation sicherzustellen, ist Softwarenutzung auf das Repertoire angewiesen, das Sie als Entwickler*in in Ihrem Produkt angelegt haben. Idealerweise sollte das zu den avisierten Zielgruppen und Nutzungssituationen passen, und viele moderne Entwicklungsmodelle, v.a. im agilen Bereich, legen darauf zurecht großen Wert. Und dennoch kommt es immer wieder zu dem, was der bekannte Soziologe Hartmut Rosa in seinem Buch „Unverfügbarkeit“ wie folgt beschrieb:

> „Immer wieder tun [Tablet oder PC] Dinge, die wir nicht verstehen, die *unlogisch*, *unmotiviert* erscheinen; sie widersetzen sich auf eine Art und Weise, die wir kaum anders denn als geradezu niederträchtig erfahren können. Es ist überaus bemerkenswert, welche Stärke unser Zorn annehmen kann, wenn sich ein Computer allen unseren Befehlen widersetzt.“ (Rosa 2019, 55; Hervorh. i.O.)

Aus der informierten Sicht von Entwickler*innen könnte eine Reaktion auf dieses Zitat die Vermutung (oder der geheime Vorwurf) sein, dass der zitierte Autor dann wohl einfach nicht wisse, wie sein Tablet oder PC funktionieren oder was er im Fall des scheinbaren Nichtfunktionierens tun müsste – quasi „RTFM: Read The Fucking Manual“ bzw. „Lies das verdammte Handbuch“, was einem zumindest in manchen Internet-Foren, in die man sich zur Fehlerklärung und Hilfesuche begibt, leider immer noch entgegengeworfen wird. So etwas ist natürlich kein Ansatz für eine beiderseitig befriedigende Kommunikation.

Dennoch ist die von Rosa geschilderte Situation so häufig wie deprimierend. Um solchen Beobachtungen entgegenzuwirken, ist es wichtig, Kommunikation zu verstehen. Eine Software, deren Aktivität als „unmotiviert“, wie Rosa schreibt, erfahren wird, ist eine Software, deren Entwickler*innen sie nicht so programmiert haben, dass ihre Ausgaben für Nutzer*innen relevante kommunikative Ausdrücke sind. Eine Software, die man als „niederträchtig“ (wieder Rosa) erlebt, ist eine, deren Entwickler*innen man unterstellt, dass ihnen die Belange der Nutzer*innen vollkommen egal sind. In der Kommunikationswissenschaft spricht man vom Kooperationsprinzip und vom Relevanzprinzip – beide Prinzipien wären hier verletzt (vgl. Kap. 2).

Wie diese und andere für Kommunikation wichtige Prinzipien zu erfüllen sind, ist eine durchaus nicht einfache Herausforderung. Software zu entwickeln, die kooperativ und relevant erscheint, geht über bloßes mechanisches Planen von Aufgabenmodellen, Nutzermodellen oder Handlungsweisen hinaus. Benötigt wird ein ganzheitliches Verständnis von Nutzungssituationen als Kommunikationssituationen. Wichtige Grundlagen hierzu möchte ich Ihnen in diesem Buch aufzeigen.

Im Buch selbst erfahren Sie alles, was Sie für einen Einstieg benötigen – ausgewählte Literaturhinweise lassen Sie optional tiefer in die jeweils behandelten Gebiete eintauchen. Dieses Buch bietet außerdem Flashcards – mit der entsprechenden iOS- und Android-App des Springer-Verlags können Sie das Erlernte festigen und anwenden üben. Und wenn Sie auch tiefer in die Hintergründe dieses Buches einsteigen wollen, werfen Sie einen Blick in die Literaturlisten der einzelnen Kapitel sowie im Kapitel „Empfehlungen zum Weiterlesen".

Ein Problem jedes Kommunikationstrainings ist meist, dass das in einem Kurs Erlernte im Alltag wieder vergessen wird. Damit das hier nicht so ist, empfehle ich Ihnen insbesondere für Kap. 1 und 2, Ihr Arbeitsumfeld als Experimentierlabor zu begreifen: Beobachten Sie sich selbst, wie Sie mit anderen kommunzieren. Beobachten Sie andere dabei. Holen Sie sich Feedback zu Ihrem Kommunikationsverhalten ein. Das kann ganz informell geschehen. Probieren Sie das Verfahren aus Kap. 3, 4, 5 und 6 testweise in kleinen Entwicklungsprozessen aus, bevor Sie es in einem großen Projekt anwenden. Diskutieren Sie mit Kolleg*innen darüber und passen Sie es für Ihre Zwecke an. Wenn Sie bei all dem offen und selbstreflexiv vorgehen, werden Sie Erfolge verzeichnen.

Berlin, Deutschland Mario Donick
30. September 2019

Literatur

Rosa, Hartmut: Unverfügbarkeit. Wien/Salzburg 2019.

Inhaltsverzeichnis

Teil I

Grundlagen – Warum es wichtig ist, Softwarenutzung zu beobachten

Einleitung: Was Software-Qualität mit menschlicher Kommunikation zu tun hat

1

Inhaltsverzeichnis

1.1 Software als Werkzeug

Es heißt, manche Programmierer*innen verstünden sich als Künstler*innen. Zumindest bauen Medien gern an diesem Bild. Im Jahr 2010 erschien in der *Computerwoche* ein Kommentar (https://www.computerwoche.de/a/programmieren-kunst-oder-handwerk,1230662), der diese Behauptung auf den Punkt brachte. Der Autor kritisierte, dass „für viele Programmierer nicht das Ergebnis im Zentrum der Überlegungen steht, sondern das Ausprobieren" – man könnte auch sagen, der Spaß am Schaffensprozess wäre das Entscheidende, aber nicht das Produkt. Das Produkt, so der Autor, wäre bestenfalls „eine schöne Zugabe".

Ähnliche Einordnungen treten immer wieder auf. Schon 1988 verglich Hans Grams in derselben Zeitschrift das Programmieren mit dem Malen eines Bildes (https://www.computerwoche.de/a/die-ekstase-des-programmierers,1155445), bei dem „der Programmierer" von einer „Faszination erfass[t]" sei, die jener „Trunkenheit, Ekstase" ähnle, die der Maler Paul Cézanne für das Malen beschrieben hatte und auf den Grams sich in seinem Artikel beruft.

Ein drittes in diese Reihe passendes Bild von Programmierer*innen als besonders herausgehobene Personen zeichnete 2002 der IT-Berater Harry Sneed, wenn auch in kriti-

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8_1

scher Absicht. Sneed behauptete, dass 75 % aller Programmierer*innen überflüssig wären. Nur wenige Entwickler*innen könnten wirklich programmieren, und in einem Projekt schleppten die wenigen guten alle anderen mit (https://www.computerwoche.de/a/die-meisten-entwickler-koennen-nicht-programmieren,531904).

Es ist bezeichnend, dass die drei genannten Sichtweisen über den Zeitraum mehrerer Jahrzehnte auftauchten und in einer Fachzeitschrift erschienen, die sich an für IT verantwortliche Führungskräfte mittlerer und großer Unternehmen richtet. Damit trug diese Zeitschrift – wie andere Medien auch – zu dem Mythos bei, den die Artikel selbst kritisieren: Softwareentwicklung scheint mindestens zu gleichen Anteilen Kunst wie Handwerk zu sein, die Kunst scheint nur von wenigen wahrhaft beherrscht zu werden, und ob das Endprodukt aus Nutzersicht gut oder schlecht ist, *scheint* eher unbedeutend.

Ob Softwareentwicklung im Ganzen oder das Programmieren als Teilgebiet nun wirklich Kunst oder eher Handwerk sind – auf jeden Fall sind sie Kulturtechniken. Sie sind von Menschen geschaffen und ihre Ergebnisse werden von Menschen genutzt. Dies kommt in der Technik-Definition des Soziologen Helmut Willke zum Ausdruck. Nach Willke bewirke Technik „eine intentional und instrumentell geschaffene Leistungssteigerung“ (Willke 2005, S. 25). Das heißt:

- Technik erlaubt es, größere Leistungen zu erbringen, als uns naturgemäß möglich ist.
- Technik wird absichtsvoll (intentional) entwickelt und verwendet, also mit einem bestimmten Ziel.
- Technik nimmt die Form von Werkzeugen an (instrumentell).

Wir nutzen also bestimmte Gegenstände, um etwas zu vollbringen, das wir ohne diese Gegenstände nicht tun könnten oder nur mit größerem Aufwand. Ein anderer Soziologe, Niklas Luhmann, hat Technik daher als „funktionierende Simplifikation“ (Vereinfachung) bezeichnet (Luhmann 1998, S. 524).

Technik, Sachtechnik und Technologie

Technik und Technologie benutzt man im Alltag oft synonym, sie sind aber nicht dasselbe.

Der Begriff *Technik* stammt aus dem Griechischen. Als „techne“ wurden Kunst, Handwerk, Kunstfertigkeiten und Können bezeichnet (Suhr 1996, S. 512); „technikos“ heißt demnach handwerklich und kunstfertig (Irrgang 2009, S. 7). In Techniksoziologie und -philosophie unterscheidet man mitunter noch Technik allgemein von *Sachtechnik*, d. h. Technik, die in Form konkreter ‚Sachen‘ (Gegenständen) auftritt. Beispielsweise ist ein Bewerbungsgespräch, an dem Sie teilnehmen, eine Technik, um herauszufinden, ob Sie zu einer Firma oder in ein Projekt passen, aber das Gespräch ist kein greifbares (berührbares) Ding. Der Stift, mit dem sich die Bewerbungskommission Notizen über Sie macht, ist das hingegen schon und damit ein Beispiel für Sachtechnik.

Abzugrenzen von Technik ist der Begriff *Technologie*. Dieser Begriff wird manchmal benutzt, um die Gesamtheit der Sachtechnik in einem bestimmten Bereich zu beschreiben, zum Beispiel im sehr weiten Begriff der Informations- und Kommunikationstech-

nologien. Der Begriff Technologie wird aber auch verwendet, um das Nachdenken über Technik und deren Folgen zu bezeichnen, so versteht etwa der Soziologe Helmut Willke Technologie als „Reflexionstheorie der Technik“ (Willke 2005, S. 128).

Ob es zu der erwünschten Leistungssteigerung kommt und ob Technik wirklich etwas vereinfacht hat, lässt sich nur anhand einer konkreten Nutzungssituation beurteilen. Erst da zeigt sich, ob die Absichten von Entwickler*innen mit den Absichten von Nutzer*innen in der Situation zusammenpassen. Damit ist es zweitrangig, welche Absichten ‚der geniale Programmierer‘ aus dem Klischee verfolgt – am Ende steht ein Ergebnis, das von anderen Menschen beurteilt wird. Bei diesem Urteil spielen die Intentionen der Entwickler*innen vielleicht gar keine Rolle – wenn sie überhaupt erkannt und verstanden werden, was u. a. am Grad der Nachvollziehbarkeit des Produkts liegt. In vorliegendem Buch gehen wir damit von folgender Grundannahme aus:

> Die Eignung und Qualität eines Werkzeugs zeigt sich erst in der konkreten, realen Nutzungssituation, aber nicht in der Entwicklungssituation, die zwangsläufig von vereinfachten Annahmen ausgehen muss.

Aus Ihrer eigenen Arbeit und Erfahrung kennen Sie natürlich ein Mittel, dieser grundlegenden Herausforderung zu begegnen, und das ist Kommunikation. Dies betrifft alle der sogenannten Stakeholder: Entwickler*innen kommunizieren untereinander, aber auch mit Auftraggeber*innen und Support-Mitarbeiter*innen, sowie mit Firmen, die als Distributor, Publisher oder Shop fungieren. Kommunikation findet weiterhin statt zwischen Nutzer*innen einerseits und Produkt, Support und Shops andererseits. Dies ist die zweite Grundannahme dieses Buches:

> Kommunikation ist das unumgängliche Werkzeug, das eine Verbindung zwischen allen an einem Produkt beteiligten oder davon betroffenen Stakeholdern, sowie zwischen Entwicklungs-, Nutzungs- und Supportsituationen ermöglicht.

Beide Grundannahmen – die Situationsgebundenheit von Software-Nutzung (und damit der Qualität von Software) sowie die Notwendigkeit von Kommunikation – werden in diesem Buch diskutiert und anhand praktischer Analyse-Beispiele demonstriert. Dabei werden wir uns einige bedeutsame Fakten bewusst machen, die man oft unhinterfragt voraussetzt oder als trivial abtun möchte, die aber große Aufmerksamkeit verdienen, wenn man Software für andere Menschen entwickelt.

1.2 Warum Software oft nicht-trivial ist und warum das eine Herausforderung ist

In Projekten arbeiten oft Menschen zusammen, die aufgrund ihrer Ausbildung, ihrer Position oder ihrer Interessen unterschiedliche Perspektiven haben und dieselben sprachlichen Begriffe unterschiedlich verstehen. Das erzeugt Missverständnisse – man verwendet viel-

leicht dieselben Worte, meint aber etwas anderes, oder man setzt ganz andere Annahmen als die Partner*innen voraus. Beispielsweise hat Niklas Luhmann einmal Computer als nicht einsehbare *Blackbox* bezeichnet (Luhmann 1996, S. 156), als schwarzen Kasten, über den wir nichts sagen können und der mit unvorhersehbaren Ergebnissen fasziniert. Aus Sicht der Informatik scheint so eine Behauptung vielleicht seltsam: Computer werden schließlich von Menschen konzipiert, entwickelt und programmiert; *selbstverständlich* können Menschen die Funktionsweise von Hard- und Software verstehen, und überraschend an den Ausgaben von Computern ist gar nichts – selbst die für Laien so erstaunlichen Resultate des Machine Learning lassen sich in ihrer Genese theoretisch rekonstruieren.

Ob man Luhmann allerdings zustimmt, ist eine Frage der Perspektive. Da schrieb ein Laie, ein Nicht-Techniker darüber, wie ihm Computer im Alltag erscheinen. Und im Alltag sind in der Tat die meisten Nutzer*innen nicht in der Lage, hinter die Oberfläche eines ihnen vorgesetzten Computerprogramms zu blicken und sich die Vorgänge und Zusammenhänge dahinter vorzustellen. Für sie ist ein Computerprogramm ein *Werkzeug*, das ‚so wie es ist' zu funktionieren hat, das aber sehr häufig gerade das nicht zu tun scheint.

Was man von Nutzer*innen (nicht) erwarten kann

Man kann heute nicht erwarten, dass Nutzer*innen wissen, was sie eigentlich alles nutzen, wozu sie das tun und welche Zusammenhänge und Abhängigkeiten zwischen einzelnen Elementen bestehen. Das Ideal scheint zurzeit im Gegenteil, dass Nutzer*innen sich gar nicht um solche Details zu kümmern sollen – ‚es' soll „einfach, schnell und sicher" (so der Werbeslogan eines bekannten Online-Finanzdienstleisters) funktionieren. Das ist das Versprechen, das hinter erfolgreichen Produkten steht.

Die aus diesem Ideal folgende Abgeschlossenheit von Software sowie ihre zunehmende Verteiltheit wird jedoch zum Problem (diesmal im negativen Sinne), wenn ‚es' doch einmal nicht wie erwartet funktioniert, also eine Störung auftritt. In der Regel sind ‚gewöhnliche' Nutzer*innen nicht in der Lage, Störungen selbst zu beheben – entweder wissen sie nicht, wie, oder sie wissen es oder können sich einen Lösungsweg herleiten, haben aber nicht die Berechtigung oder nicht die technischen Mittel, die Lösung auch umzusetzen.

Daraus entsteht natürlich ein Markt für Unterstützungsleistungen des technischen Supports, doch das mangelnde Bewusstsein über Funktion und Zusammenhänge bei den meisten Nutzer*innen ist auch dann eine große Herausforderung, wenn es darum geht, sie im Störungsfall zu unterstützen – einmal ganz abgesehen von lauter werdenden Forderungen, nach denen Nutzer*innen eine „Code Literacy" (Rushkoff 2010) entwickeln sollten, damit sie im Angesicht übermächtiger Technologiekonzerne und an Überwachung interessierter Staaten weiterhin mündige Bürger*innen bleiben (dazu mehr in Abschn. 1.5).

Das Werkzeug *nutzen* Menschen, um ein *Problem* zu lösen, das heißt einen „gegebenen Zustand in den erwünschten Endzustand zu transformieren" (Dörner 1984, S. 11). Das Werkzeug hilft bei der Problemlösung, wenn dank ihm weniger Operationen zur Lösung

des Problems nötig sind, wenn es die Ausführung dieser Operationen vereinfacht oder wenn es Zugriff auf ein irgendwo abgelegtes Lösungsschema bietet. Beispielsweise ist ein Übersetzungsproblem von einer Sprache in eine nur selten genutzte andere Sprache heute mit Online-Übersetzern meist schneller in ausreichender Weise gelöst, als manuell mit Nutzung von Wörterbüchern, Grammatiken und mühsam erinnertem Fremdsprachenunterricht.

Der Begriff *Nutzung* meint – anders als bei einfachen Werkzeugen (wie Hammer, Bohrmaschine, usw.) – nicht nur die bewusste Anwendung der Software, d. h. den aktiven Entschluss von Nutzer*innen, ein Programm zu starten und zu bedienen (etwa eine Textverarbeitung oder ein Computerspiel). Darüber hinaus meint Nutzung in diesem Buch gerade auch Software, die von Nutzer*innen unbemerkt aktiv ist, mitunter auch ungewollt, nichtsdestotrotz aber die Lösung der jeweiligen Probleme beeinflusst (etwa die Hintergrunddienste eines Betriebssystems oder Software, die auf Servern läuft, auf die Nutzer*innen von zu Hause aus zugreifen).

Bei der Problemlösung wird ein Eingangswert unter Nutzung bestimmter Verfahren in einen Ausgangswert transformiert. Bei softwaregestützter Problemlösung kann man von einer *Maschine* sprechen. Diese Maschine erscheint nach außen hin als Gefüge aus physisch vorhandener Hardware und programmierter, Daten verarbeitender Software – der Computer. Schaut man von außen, als Nutzer*in, auf eine Maschine, kann sie als *triviale* Maschine oder als *nicht-triviale* Maschine erscheinen (vgl. von Foerster 1993, Abb. 1.1 und 1.2).

Diese Unterscheidung wurde vom Physiker Heinz von Foerster (1911–2002) eingeführt, um unterschiedlich komplexe Maschinen zu beschreiben:

- Eine triviale Maschine entspricht der Funktion $y = f(x)$. Bei zwei Durchläufen der Maschine kommt bei gleichem Eingangswert x stets derselbe Ausgangswert y heraus (von Foerster 1993, S. 246).
- Eine nicht-triviale Maschine entspricht der Funktion $y = f(x, z)$, wobei z ein von außen nicht beobachtbarer Zustandswert in der Maschine ist, der die Berechnung von y beeinflusst. Bei zwei Durchläufen der Maschine kann bei gleichem Eingangswert x doch ein anderer Ausgangswert y herauskommen, je nach Wert von z (ebd., S. 247 f.). Wenn man den inneren Aufbau der Maschine nicht kennt und daher nicht weiß, wie z zustande kommt, kann es sehr schwer sein, das Ergebnis zu erklären.
- Bei einer rekursiven nicht-trivialen Maschine wird der Ausgangswert y als neuer Eingangswert x in einen weiteren Durchlauf der Maschine eingeführt (ebd., S. 362).

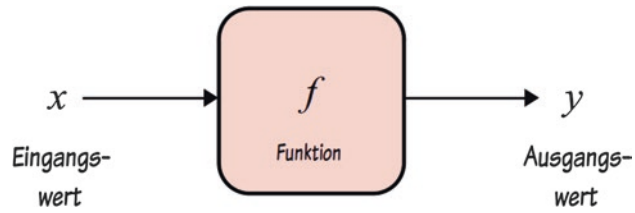

Abb. 1.1 Triviale Maschine nach Heinz von Foerster

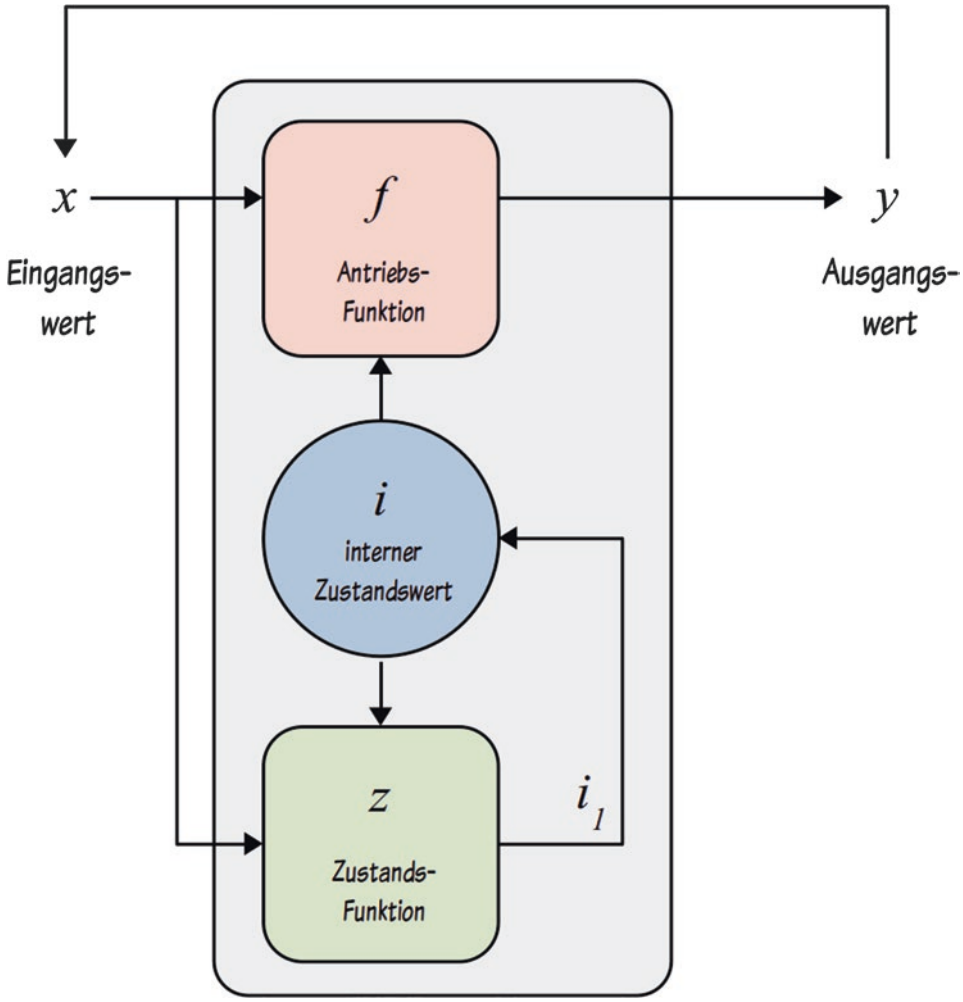

Abb. 1.2 Nicht-triviale Maschine nach Heinz von Foerster

Endliche Zustandsautomaten und (nicht-)triviale Maschinen

In der Informatik spricht man selten von Heinz von Foersters Unterscheidung trivialer und nicht-trivialer Maschinen. Das liegt wohl daran, dass diese Unterscheidung aus technischer Sicht selbst eher trivial ist – von Foerster beschreibt nichts weiter als endliche Zustandsautomaten. Endliche Automaten sind formale Modelle, mit denen die Zustände mehr oder weniger komplexer Systeme beschrieben werden, zum Beispiel eine Kundin, die etwas kauft und per Banküberweisung bezahlt (vgl. Hopcraft et al. 2002, S. 48).

Man unterscheidet deterministische und nicht-deterministische endliche Automaten (DEAs und NEAs; ebd., S. 54 ff.). Beide Typen haben eine endliche Menge möglicher

Zustände Q, eine endliche Menge möglicher Eingangssymbole $\sum$ (Eingangswerte) und eine endliche Menge möglicher finaler/akzeptierender Zustände F (als Teilmenge von Q; Ausgangswerte, ebd.). Um von den Eingangswerten auf die Ausgangswerte zu kommen, gibt es eine Übergangsfunktion. Die Übergangsfunktion eines DEA gibt genau einen Zustand zurück; die Übergangsfunktion eines NEA kann keinen, einen oder mehrere Zustände zurückgeben (ebd., S. 65).

Die Übergangsfunktion ist bei von Foerster in Antriebs- und Zustandsfunktion geteilt. Heinz von Foerster will damit auf die Frage hinweisen, ob die Zustände des Automaten und Berechnung der Ausgangswerte *analytisch determinierbar* sind (von Foerster 1993, S. 251) – das ist nicht dasselbe wie deterministisch, und darin liegt die Wichtigkeit von Foersters Konzept für uns: Ein Automat ist analytisch determinierbar, wenn die Zustände des Automaten von außen (durch einen menschlichen Beobachter) bestimmbar oder vorhersehbar sind, ohne dass dieser Mensch über den inneren Aufbau des Automaten Bescheid weiß.

Eine triviale Maschine ist damit ein Automat, der deterministisch und determinierbar ist; eine nicht-triviale Maschine dagegen ein Automat, der zwar deterministisch, aber nicht determinierbar ist. Sowohl triviale als auch nicht-triviale Maschinen sind deterministisch, da sie einen Zustand oder Ausgangswert zurückgeben. Die Nicht-Trivialität liegt in der Übergangsfunktion, die schon bei DEAs so komplex sein kann, dass ‚normale Nutzer*innen' die möglichen Zustände und Ausgangswerte nicht antizipieren können.

Von Foerster bezieht seine Unterscheidung trivialer und nicht-trivialer Maschinen nur auf DEAs, aber für NEAs gilt sie umso mehr, womit wir hier über von Foersters Einschränkung auf DEAs hinausgehen. NEAs erscheinen aus Sicht ‚unbedarfter' Beobachter *immer* nicht-trivial, aber auch DEAs können nicht-trivial erscheinen, wenn die Übergangsfunktion zwar nur einen Wert zurückgibt, es aber trotzdem eine größere Menge von Zuständen gibt. Des Weiteren kann auch das Zusammenspiel mehrerer trivialer DEAs zu einer Ordnung führen, die dann selbst als nicht-triviale Maschine beobachtbar ist (vgl. Donick 2019, S. 117 ff.).

Wegen der häufigen Nicht-Trivialität endlicher Automaten nutzt man während Planung und Entwicklung solcher Automaten oft Hilfsmittel wie Übergangstabellen und Übergangsgraphen. Diese Werkzeuge helfen dabei, während der Entwicklung den Überblick zu behalten und nicht selbst von Nicht-Trivialität überfordert zu werden. Nutzer*innen hingegen haben diese Möglichkeit in der Regel nicht, und das kann zu Nutzungsschwierigkeiten führen, wenn die Ausgangswerte einer nicht-trivialen Maschine nicht zu den Erwartungen der Nutzer*innen passen.

Von Foersters Beschreibung geschieht in Hinblick auf Beobachtung. Bestenfalls schauen wir als aktive Nutzer*in, schlimmstenfalls als betroffene Unbeteiligte von außen auf eine Maschine. Diese Maschine mag aus Entwickler*innen-Sicht gar nicht sonderlich komplex sein, aber dennoch kann sie aus Sicht ‚normaler Leute' nicht-trivial wirken, weil diese ‚normalen Leute' nicht wissen und nicht wissen können, was im Inneren der Ma-

schine vor sich geht – darum geht es. Mit von Foerster ist eine nicht-triviale Maschine zwar *deterministisch, aber analytisch nicht determinierbar* (vgl. von Foerster 1993, S. 250 ff.) – die Maschine hat eine endliche Menge von Zuständen und ist theoretisch berechenbar, aber praktisch ist letzteres nicht möglich.

Die große Bedeutung des Konzepts der nicht-trivialen Maschine liegt in der Klarheit, mit der darin das Beobachterproblem ausgedrückt wird, das menschlichem Zusammenleben zugrunde liegt. Die tatsächliche Leistung von Technik für ihre Nutzer*innen kann nur anhand der Beobachtung dieser Nutzer*innen beurteilt werden. Die Leistung von Technik kann von ihren Entwickler*innen nicht vollständig kontrolliert werden, und mitunter ist die tatsächliche Leistung sogar eine ganz andere als die bei der Entwicklung beabsichtigte (z. B. bei Zweckentfremdung von Technik).

Wir sind nur Beobachter*innen

Als Menschen können wir immer nur von außen, als Beobachter*innen, auf andere Akteure blicken. Trotzdem ist erstaunlicherweise Verständigung möglich. Der Soziologe Niklas Luhmann sprach von der Unwahrscheinlichkeit von Kommunikation (Luhmann 1992). Luhmann hat eine sehr umfangreiche soziologische Systemtheorie entwickelt, in der er alles konsequent als Systeme beschreibt (vgl. Luhmann 1992, 1996, 1998).

Psychische Systeme (Bewusstseine), soziale Systeme (Paare, Teams, Gesellschaften, …), das Wirtschaftssystem, das politische System, das religiöse System, das System der Massenmedien, das Wissenschaftssystem usw. können sich zwar gegenseitig beobachten, und sich auch beim Beobachten beobachten (das sind dann sogenannte Beobachter 2. Ordnung), aber nichts über den tatsächlichen Zustand eines beobachteten Systems wissen.

Damit zwischen Systemen dennoch Austausch geschehen kann, können Systeme aneinander strukturell gekoppelt sein; sie können dann Leistungen füreinander erbringen. Systeme können auch durch Umwelteinflüsse irritiert (gestört) werden und müssen sich dann ggf. ausdifferenzieren (Evolution), um mit der Irritation zurechtzukommen. Was aber wirklich im Innern eines beobachteten, gekoppelten oder möglicherweise irritierten Systems vor sich geht, kann ein anderes System nicht bestimmen.

Wenn Luhmann also einen Computer als *Blackbox* bezeichnet, dann betrachtet er diesen Computer als nicht-triviale Maschine, dem zumindest Laien nicht ansehen und für den sie sich auch nicht herleiten können, wie er – einzeln oder vernetzt – zu seinen Ergebnissen kommt – ganz so, wie wir auch andere Menschen nur von außen beobachten können. Luhmann beschreibt alles als Systeme in einer Umwelt, und die Systeme können sich nur beobachten, und nur beobachten, wie sie beobachten. Auch Technik ist für Menschen Umwelt, die wir in Gestalt und Aktivität zuallererst beobachtend wahrnehmen (Abb. 1.3).

Der Begriff der nicht-trivialen Maschine erlaubt es, die Problematik solcher Beobachtungsvorgänge prägnant zu bezeichnen. Daher ist die nicht-triviale Maschine nicht nur im engen Sinne als sogenannte *Sachtechnik* zu verstehen (d. h. als physisch vorhandene Sa-

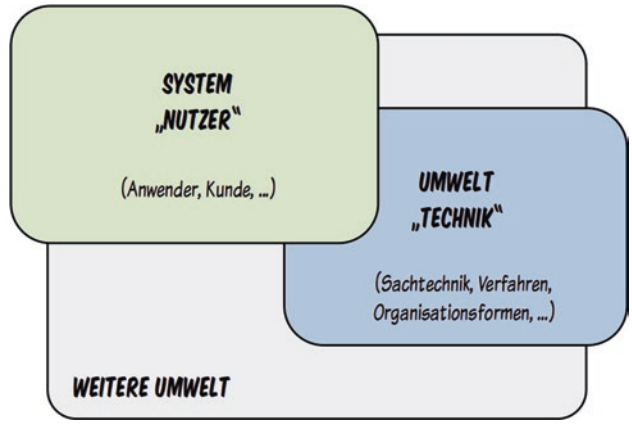

Abb. 1.3 Techniknutzung als Beziehung von System und Umwelt

che, etwa Hardware und in irgendeiner Weise physisch gespeicherter und verarbeiteter Software). Die nicht-triviale Maschine geht darüber hinaus. Sie ist zu verstehen als Sachtechnik, deren Leistung erst vor dem Hintergrund zahlreicher anderer Parameter der Nutzungssituation einzuschätzen ist.

1.3 Die Bedeutung der Nutzungssituation

1.3.1 Situation, Handlung und Verhalten

Technik wird in bestimmten Situationen entwickelt und genutzt. Eine Situation ist durch eine Vielzahl von Parametern gekennzeichnet, die während der Entwicklung von Technik nicht vollständig vorhersehbar und damit auch nicht alle zu berücksichtigen sind. Dennoch gibt es zahlreiche sogenannte Kontextmodelle unterschiedlicher Granularität, die genau das versuchen (vgl. Kasten). Zur Situation (oder zum Kontext) gehören etwa die menschlichen Nutzer*innen und ihre Vorlieben, andere betroffene Menschen, verschiedene örtliche und zeitliche Gegebenheiten (z. B. die konkrete Arbeitsumgebung, aber auch der Ort als kultureller Hintergrund), organisationale Faktoren (z. B. Hierarchien, Teamstrukturen), Prozesse und andere Sachtechnik (z. B. beim Datenaustausch zwischen zwei Programmen). All diese Parameter tragen zum z in von Foersters Definition der nicht-trivialen Maschine bei, können es aber nicht vollständig beschreiben.

Situation und Kontext

Menschen handeln in konkreten Situationen. Dasselbe Werkzeug kann in unterschiedlichen Situationen eingesetzt werden und dann auch unterschiedlich wirksam sein. Als *Situation* werden im Duden „Verhältnisse, Umstände" bezeichnet, „die einen allgemeinen Zustand definieren". Als Individuen haben wir nie eine identische Situationsdefinition, sondern unterschiedliche subjektive Repräsentationen von Situationen. Je nach erkenntnistheoretischer

Position können uns diese Unterschiede als mehr oder minder stark ausgeprägt erscheinen (vgl. Donick 2016, S. 69 f.). Zum *Kontext* im soziologischen Sinne kann man die Elemente einer Situation bezeichnen, die unser Handeln in dieser Situation bestimmen.

Um Software als Werkzeug möglichst situationsgerecht zu gestalten, werden Kontextmodelle verwendet. In einem Kontextmodell werden messbare Elemente einer Situation modelliert, die man als handlungsleitend für die Situation annimmt. Häufig bezieht sich die Informatik auf die Kontextdefinition von Anind Dey und Gregory Abowd. Danach ist Kontext „any information that can be used to characterize the situation of an entity" (Dey und Abowd 1999, S. 3 f.). Der in der Definition benutzte Begriff „Entität" (entity) weist darauf hin, dass nicht nur Menschen einen Kontext ‚haben' können, sondern auch Technik, wie eben Software. Erst dadurch kann ein Informationsbegriff genutzt werden, der Information nicht als psychische Repräsentation versteht, sondern als technisch erfassbare Elemente. Diese Informationen sind, sofern mess- und modellierbar, Kontext der Entität. Im Laufe der Zeit wurden zahlreiche Kontextmodelle entwickelt (vgl. die Vergleichsstudie von Bolchini et al. 2007), die auf der Definition von Dey und Abowd aufbauen oder einen ähnlichen Ansatz vertreten.

Aus kommunikationssoziologischer Sicht ist der Kontextbegriff der Informatik zu eingeschränkt. Kontext als Menge der handlungsleitenden Elemente einer Situation umfasst mehr als die mess- und modellierbaren Elemente der Situation. Aufgrund des Beobachterproblems gehören zum Kontext auch Annahmen, die Menschen über die Situation haben, und Meta-Annahmen, d. h. Annahmen über die Annahmen anderer Menschen und Entitäten der Situation (vgl. Donick 2016, S. 72 ff.; Sperber und Wilson 1995, S. 15 f.). Solche Elemente sind nur schwer modellierbar, müssen aber trotzdem bei der Entwicklung von Software berücksichtigt werden, wenn die Software situationsgerecht sein soll.

Das liegt auch an den Verhaltens- und Handlungsweisen der Menschen einer Situation. Der Unterschied von Verhalten und Handeln liegt in der schon erwähnten Absicht (Intention) der Menschen. Liegt der Aktivität eines Menschen eine bewusste Intention zugrunde, spricht man von Handeln; die Person verfolgt eine Absicht, hat ein Ziel, vielleicht auch einen Plan, den sie schrittweise umsetzt. Geschieht eine Aktivität ohne bewusste Intention, spricht man stattdessen von Verhalten (vgl. Linke et al. 2004, S. 197 f.). Da sich Menschen nicht immer über ihre Absichten im Klaren sind, kann die Grenze zwischen Handeln und Verhalten mitunter schwer zu bestimmen sein.

Der für die Entwicklung von Technik relevante Punkt ist, dass Menschen auch bei der Nutzung von Technik nicht immer bewusst handeln – die oben gezeigte Aufzählung, nach der Technik absichtsvoll und mit einem bestimmten Ziel verwendet wird, war tatsächlich idealisierend. Gerade bei Softwarenutzung verwenden wir oft Technik, ohne uns bewusst dafür entschieden zu haben oder ohne genau zu wissen, was wir eigentlich tun:

- Wir nehmen die Programme, die vorinstalliert sind oder
- wir verwenden dasjenige, was uns die *Google*-Suche oder der *App Store* als erstes vorschlägt;

- mit dem eigentlichen Werkzeug werden uns weitere Programme aufgenötigt, deren Rolle uns unklar bleibt;
- wir klicken auf der Suche nach einer Programmfunktion unsystematisch umher, bis ‚es irgendwie geht';
- wir ‚wursteln' uns mangels Transparenz der Software und Kenntnis ihres Aufbaus mitunter umständlich durch, statt klare Entscheidungen zu treffen;
- wir wissen bei Störungen oft nicht, was wir selbst tun können oder tun sollten.

Bei der Softwarenentwicklung erstellte Aufgabenmodelle, die versuchen, erwünschtes oder wahrscheinliches Nutzerhandeln wie einen Algorithmus zu modellieren, darauf aufbauend die grafische Benutzeroberfläche (GUI) einer Software zu gestalten und die Effizienz der Bedienung dieser GUI zu bewerten, hängen einem Ideal stets absichtsvoll handelnder und konzentrierter Nutzer*innen an. Dieses Ideal ist unrealistisch. Aufgabenmodelle sind genauso verkürzend wie es Kontextmodelle für die Modellierung von Situationen sind. Es ist wichtig anzuerkennen, dass die Ungewissheit menschlichen Nutzerhandelns und -verhaltens keinen bloßen Mangel an Ausbildung der Nutzer*innen darstellt, der durch bessere Ergonomie, eine bessere GUI, bessere Handbücher oder mehr Schulungen vollständig zu beheben wäre. Ungewissheit gehört zum Menschsein und damit zum Umgang mit Technik dazu, und Technik muss darauf eingerichtet sein.

Aufgabenmodelle als eingeschränkte Modelle menschlichen Handelns

Ähnlich wie Kontextmodelle handlungsleitende Elemente von Situationen technisch zu erfassen versuchen, so nutzt man Aufgabenmodelle wie GOMS (Card et al. 1983), NGOMSL (Kieras 1996) oder Werkzeuge wie CogTool, um menschliches Handeln technisch zu modellieren. Dies ist nicht nur ein Hilfsmittel für die Entwicklung, sondern soll auch quantitative Vorhersagen darüber ermöglichen, wie effizient die jeweilige Software sein wird. Da empirische Usability-Tests zu zeitaufwendig und teuer für die praktische Softwareentwicklung seien (so etwa Kieras 1996 in seiner Einführung zur Modellierungssprache NGOMSL), könne man stattdessen Aufgabenmodelle nutzen, um die Qualität eines Designs vorherzusagen.

Den üblichen Aufgabenmodellen ist gemein, dass sie menschliches Handeln quasi algorithmenhaft beschreiben. Sie gehen davon aus, dass Handeln immer einem Plan folgt; dass Handeln immer in einzelne Schritte zerlegbar ist und dass es unterschiedlich effizientes Handeln geben kann, wobei Effizienz oft in zeitlicher Hinsicht und als ‚möglichst wenig Nutzereingaben' verstanden wird. Diese Sicht erinnert an klassische Problemlösetheorien, ist aber aus kommunikationssoziologischer Perspektive nicht durchhaltbar. Wie schon Suchman (2007) gezeigt hat, weichen Nutzer*innen häufig von eigenen oder durch Entwickler*innen vorgegebenen Plänen ab. Daher können Aufgabenmodelle, genau wie Kontextmodelle, bestenfalls Annäherungen an eine Nutzungssituation sein, diese aber nie hinreichend beschreiben. Um Software situationsgerecht zu gestalten, muss man mit Abweichungen vom Modell rechnen bzw. in der Software auch nichtmodellierbare Nutzungswege und -alternativen vorsehen.

Zusammengefasst ist bis hierher festzuhalten:

> Eine Software wird in einer umfassenden Situation eingesetzt, die mehr als eine Konstellation messbarer oder modellierbarer Parameter ist. In der Situation wird Software bewusst oder unbewusst von Menschen verwendet, die nicht immer absichtsvoll und rational handeln.

Die Leistung von Software kann nur in diesen Situationen oder in deren Nachgang durch beobachtende Menschen beurteilt werden. Das aber heißt:

> Die Qualität von Software ist ein situationsabhängiger Wert, der mit Ungewissheit rechnen muss.

Letztlich geht es in vorliegendem Buch also darum, wie Sie unter diesen unklaren Bedingungen trotzdem qualitativ hochwertige Software entwickeln können, was ‚qualitativ hochwertig' überhaupt bedeutet und wie Sie ermitteln, ob Sie erfolgreich waren.

1.3.2 Softwarequalität als situationsabhängiger Wert

Wenn wir das im vorigen Abschnitt gezogene Zwischenfazit akzeptieren, ist die Frage, wie wir damit praktisch umgehen. Wenn uns Messungen und Modellierungen nur bedingt helfen können, situationsgerechte Software zu entwickeln, müssen wir dann nicht einfach mit der verbleibenden Unschärfe leben und – provokant gefragt – dann eben erst bei in der Situation auftretenden Nutzungsproblemen aktiv werden? Ist es nicht sinnvoller, unser Produkt so gut es eben geht zu gestalten und um Probleme kümmern wir uns (oder noch besser: der Support, vgl. Donick 2019) hinterher?

Das stimmt. Und auch, wenn Sie dieses Buch zu Ende gelesen haben, wird sich daran nichts grundlegend ändern. Wir können aber die Fallhöhe reduzieren, die zwischen einer Konstellation von modellierbaren Parametern einerseits und nicht messbaren situativen Faktoren andererseits liegt und so die Unschärfen reduzieren. Es ist sogar möglich, Softwarequalität als situationsabhängigen Wert zu definieren und Annahmen zu entwickeln, wie die Qualität einer bestimmten Software in möglichen Situationen vermutlich sein wird. Wie kann so eine Definition aussehen?

Eine übliche Definition von Qualität liefert die ISO-Norm 8402, die Qualität als „Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit" bezeichnet, „die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht." (Zuser et al. 2001, S. 284). Diese Eigenschaften und Merkmale sollten messbar und/oder beobachtbar sein (ITU 2008, S. 3). Ein Versuch, situative Ungewissheit und Messbarkeit zu vereinen, ist das Konzept der Quality of Interaction (QoI, Interaktionsgüte, vgl. Daher et al. 2010, 2012). QoI betrachtet technische und menschliche Parameter gemeinsam, um die Interaktion mit und vermittels technischer Informations- und Kommunikationssysteme zu bewerten.

QoI ist als Ergänzung zu Quality of Service (QoS, Dienstgüte) und Quality of Experience (QoE, Erfahrungsgüte oder besser -qualität) gedacht. Während QoS technische Ressourcen

und Leistungsparameter bei der Datenübertragung betrachtet (vgl. grundlegend ITU 2001, 2008), und bei QoE die subjektive Bewertung einer Situation durch ein Individuum im Fokus steht (vgl. Killki 2008), soll QoI die „tatsächliche Interaktionsgüte“ (Donick et al. 2012, S. 108) bei computervermittelter Kommunikation zwischen Menschen oder zwischen Menschen und Computern messen. Das „soll die Frage beantworten, wie effizient und erfolgreich die zur Verfügung stehenden Ressourcen auf Anwendungsebene […] von menschlichen Nutzern zur Lösung ihrer jeweiligen Aufgaben“ (ebd.) verwendet werden.

QoI lässt sich im Rahmen des Verfahrens ermitteln, das in vorliegendem Buch in Teil II beschrieben wird. Von QoI ausgehend, lassen sich dann Anknüpfungspunkte an etablierte Bewertungskriterien für Softwarequalität finden, etwa die Norm DIN EN ISO 9241-110 („Grundsätze der Dialoggestaltung“), mit der die Ergonomie von Software beschrieben wird. Wie das genau funktionieren kann, schauen wir uns in Kap. 6 des Buches an. Relativ ungewohnt ist an dem dort diskutierten Ansatz, dass die ermittelten Werte nicht absolut sind, sondern eine Bezugsnorm abstecken, die je nach Situation interpretiert wird. Damit soll die Ungewissheit menschlicher Kommunikation berücksichtigt werden.

1.4 Menschliche Kommunikation bei Software-Entwicklung und -Nutzung

Wenn Sie an der Entwicklung von Software beteiligt sind, die von anderen Menschen genutzt werden soll, dann sind sie gezwungen, mit anderen Menschen zu kommunizieren. Je nach Ihrer Rolle unterscheiden sich die Kommunikationssituationen, aber an irgendeiner Stelle will jemand etwas von Ihnen, oder Sie wollen etwas von jemandem. Dies betrachten wir in Kap. 2 des vorliegenden Buches genauer, aber bitte denken Sie schon jetzt einmal kurz über die folgenden Fragen nach:

1. Sind Sie eher eine Person, die – projektbezogen – gerne und viel kommuniziert, oder haben Sie den Eindruck, dass viel zu viel geredet wird und man mehr arbeiten sollte?
2. In welchen Situationen bei der Software-Entwicklung kommunizieren Sie persönlich mit anderen Menschen? In welchen davon *wollen* Sie das, in welchen davon sind Sie dazu gezwungen?
3. Welche der Situationen, die Sie in Frage 2 gefunden haben, kommen in Ihrem Alltag am häufigsten vor?

Ich vermute, Sie haben eine ganze Menge verschiedener Situationen zusammengetragen. Hier einmal eine Auswahl, die mir spontan beim Schreiben in den Sinn kam:

- Sie verhandeln mit potenziellen Auftraggebern oder stellen – auf der Suche nach Förderung – Ihr Projekt in einem Pitch vor;

- Sie entscheiden sich für ein Entwicklungsmodell und müssen dafür sorgen, dass dem Modell entsprechend gearbeitet wird;
- Sie sitzen stundenlang in Meetings und Videokonferenzen, bekommen Anweisungen von Ihren Vorgesetzten, oder Sie geben selbst welche;
- Sie schreiben Kommentare in einen Quelltext oder sollen aus den Kommentaren eines anderen Quelltexts schlau werden – oder Sie arbeiten sich in fremden Quelltext ein, der so ganz anders ist als Sie ihn schreiben würden, und dann ist er auch noch unkommentiert;
- Sie erzeugen eine Dokumentation für Ihr Projekt oder lesen die eines anderen;
- Sie gestalten Benutzeroberflächen, die von späteren Nutzer*innen verstanden werden müssen: grafische Elemente, Animationen, Klänge, Menü-Einträge, Fehlermeldungen usw.;
- Sie schreiben Handbücher, Hilfetexte, Tutorials usw. für die Nutzer*innen Ihres Produkts;
- Ein*e Nutzer*in versucht, mit den von Ihnen produzierten Medien (Benutzeroberfläche, Funktionsumfang, Handbücher usw.) zurechtzukommen.

Ich denke, die meisten Beispiele werden Sie in der einen oder anderen Form nachvollziehen können, je nachdem, was Ihre Rolle ist. Erfahrungsgemäß ernte ich aber Verwunderung beim letzten Punkt – denn der ist offensichtlich kein Bestandteil der Entwicklungssituation mehr, sondern gehört zur Nutzungssituation, und Sie haben damit nichts mehr zu tun (außer Nutzer*innen fragen Sie aktiv um Hilfe). Warum zähle ich die Nutzung hier als Beispiel für Kommunikation bei der Software-Entwicklung auf?

Weil Nutzer*innen während der Nutzung Annahmen über Sie als Entwickler*in bilden. Diese Annahmen sind oft weniger explizit wie es Ihre Annahmen über Nutzer*innen sind. Als Entwickler erstellen Sie mitunter nicht nur Kontextmodelle und Aufgabenmodelle, sondern auch Nutzermodelle – Beschreibungen Ihrer künftigen Nutzer*innen. Daran richten Sie Ihr Produkt aus (außer, Sie verstehen sich doch eher als Künstler*in und richten Ihr Programm vorwiegend an Ihren eigenen Vorstellungen aus). Solche Modelle entwickeln Nutzer*innen nicht (warum sollten sie auch). Dennoch sind die Annahmen da – sie werden zum Beispiel dann manifest, wenn Nutzer*innen über Sie ‚schimpfen':

- „Was haben die sich nur dabei gedacht, dass die in der neuen Programmversion die ganze Benutzeroberfläche verändert haben?"
- „Die haben offensichtlich keine Ahnung vom Arbeitsalltag, in dem ich gezwungen bin, diese Software einzusetzen."
- „Statt so viel Zeit in neue Features zu stecken, sollten die lieber erst mal die bestehenden Fehler reparieren!"

Solche und ähnliche Äußerungen kommen immer wieder vor. Nur die wenigsten Personen machen sich die Mühe, sie Ihnen als Feedback über Kontaktformulare oder in Foren-Communitys mitzuteilen, aber darum geht es hier nicht. Für uns ist wichtig, dass hinter schimpfenden Nutzer*innen oft das Problem steht, dass diese Ihre Absichten nicht verstehen. Sie können zwar die Bedienung Ihres Programms erlernen – mechanisch, ein-

fach weil es nicht anders geht –, aber wenn sie nicht nachvollziehen können, warum es in der oder der Weise bedient werden soll, oder wie die Ausgaben des Programms zu verstehen sind, werden sie nicht zufrieden sein.

Deshalb sind Sie als abwesender Entwickler Bestandteil der Nutzungssituation. Ihre Software wird mindestens unbewusst als kommunikative Äußerung verstanden, die Sie den Nutzer*innen gegenüber tätigen. Wenn es um Kommunikation geht, wird gern folgende Aussage zitiert: „Man kann nicht nicht kommunizieren“. Dies stammt vom bekannten Psychologen Paul Watzlawick (vgl. Watzlawick et al. 2016 und Kasten in Kap. 2). Das heißt: Es ist nicht wichtig, ob Sie kommunizieren *wollen*, sondern ob man Ihnen anhand Ihres Handelns, Ihres Verhaltens oder anhand von Ihnen produzierter Medien *unterstellt*, Sie würden etwas kommunizieren. Dabei sind „Sie“ in einer Software-Nutzungssituation in der Regel nur als mehr oder minder diffuse Vorstellung der Nutzer*innen vorhanden, die zwar keine konkrete Person beschreiben kann, aber der man trotzdem alle möglichen Eigenschaften zuschreibt, je nachdem, welche Erfahrungen und Vorurteile man als Nutzer*in mit Ihrem Beruf verbindet – hier kommt wieder das zu Beginn des Kapitels erwähnte Bild ins Spiel, das in Massenmedien über Programmierer*innen, Programmierung und Software-Entwicklung konstruiert und verbreitet wird.

Bitte denken Sie einmal über folgende Fragen nach:

1. Stellen Sie sich die Benutzeroberfläche eines von Ihnen entwickelten Programms vor. Finden Sie zwei oder drei Beispiele für Elemente dieser Oberfläche, anhand derer sich Nutzer*innen (möglicherweise unbeabsichtigte) Vorstellungen über Sie als Programmierer*in machen könnten.
2. Wie könnten Sie verhindern, dass Nutzer*innen falsche Annahmen über Ihre vorhandenen (und nicht vorhandenen) Absichten entwickeln?

Hier drei Beispiele für die erste Frage:

- Es kann sein, dass manche Personen mitunter das Gefühl haben, Sie würden deren Vorlieben nicht respektieren („Unmöglich. Schon wieder ein Programm, das eigene Farben nutzt statt die, die ich systemweit eingestellt habe!“)
- Es kann auch sein, dass man Ihre Entscheidung als unpassend empfindet („Ernsthaft? Die nutzen in dem Spiel wirklich Arial als Standardschriftart? Langweiliger geht es ja wohl nicht.“)
- Und schließlich kann es auch sein, dass man sich einfach nur fragt, was das Programm da eigentlich die ganze Zeit tut („Hm, komisch, der Fortschrittsbalken steht jetzt seit zehn Minuten bei 100 % und trotzdem geht es nicht weiter. Was ist das los?“)

In diesen Beispielen verstehen die Nutzer*innen nicht, was Ihre Absicht war – warum nehmen Sie nicht das Farbschema, das im Betriebssystem eingestellt ist? Warum verwenden Sie für Ihr ansonsten so hochwertig aussehendes Science-Fiction-Spiel Arial als

Schriftart? Warum ist der Fortschrittsbalken so konfiguriert, dass der angezeigte Prozentwert offenbar nicht der tatsächlichen Aktivität des Programms entspricht?

Es muss Nutzer*innen während der Nutzung deutlich werden, warum Ihre Entscheidungen sinnvoll waren. Wenn es das nicht wird, dann sticht die Entscheidung hervor und wird hinterfragt. Sie ist dann kommunikativ wirksam. Im schlimmsten Fall kann das die Nutzung stören und zu Unzufriedenheit führen.

Allerdings wäre es kaum sinnvoll, würden Sie Ihre Entscheidung irgendwo erklären – selbst wenn Sie das täten (z. B. in einem Entwickler-Tagebuch, wie es sie heute oft gibt), läse das nur eine Minderheit. Die Nutzung selbst muss insgesamt konsistent (oder kohärent, vgl. Kap. 5) erscheinen; Ihr Programm muss aus sich selbst heraus ‚rund' wirken. Es sollte also keine Momente geben, in denen Nutzer*innen aufgrund irgendwelcher ungewollt kommunikativ wirksamer Elemente stutzig werden.

Um dies zu erreichen, können Sie Software transparent gestalten – aber hier ist mit Transparenz nicht gemeint, so viel wie möglich vor Nutzer*innen zu verstecken (damit diese sich nicht mit Komplexität auseinandersetzen müssen), sondern im Gegenteil die Software so zu gestalten, dass Nutzer*innen jederzeit wissen, was die Software gerade tut und aus welchem Grund, und wie ihre Ergebnisse zustande gekommen sind.

1.5 Transparente Software als gesellschaftliches Ideal

Aus der Situationsabhängigkeit der Nutzung einerseits und der Rolle der menschlichen Kommunikation für Entwicklung und Nutzung andererseits lässt sich eine Forderung nach *Transparenz* ableiten. Transparenz ist hier nicht im manchmal genutzten technischen Sinne der Unsichtbarkeit gemeint (nach der ein Computersystem zwar vorhanden ist, aber nicht bemerkt werden soll), sondern als dessen Gegenteil:

> Transparent gegenüber Nutzer*innen ist Software, wenn Nutzer*innen jederzeit die Möglichkeit haben, sich über Hintergründe und Aktivitäten der Software in der aktuellen Situation sowie über das Zustandekommen der Berechnungs- oder Verarbeitungsergebnisse zu informieren.

Die undurchsichtige Blackbox Software soll also ein Stück weit transparent gemacht werden. Hierbei kann es um ganz grundlegende Dinge gehen, die Nutzung erst ermöglichen oder verhindern. Beispielsweise fragen Smartphones und Tablets mit Android-Betriebssystem bei der erstmaligen Nutzung einer neuen App häufig nach Berechtigungen für die App. Den Nutzer*innen werden Hinweise angezeigt, dass die App etwa die Kamera benutzen, auf den Speicher zugreifen oder Telefonanrufe tätigen will, und das kann man erlauben oder verhindern. Manchmal erklärt sich die Notwendigkeit für eine Berechtigung von selbst, aber oft kann man als Nutzer*in nicht erkennen, wozu eine App eine bestimmte Berechtigung braucht. Beliebt sind etwa Taschenlampen-Apps, die das eingebaute Blitzlicht des Geräts verwenden. Aus Nutzer*innen-Sicht haben die eine ganz simple Funktion, nämlich ‚Licht an, Licht aus'. Da aber das Blitzlicht im Betriebssystem über die Kamera-

Schnittstelle (API) gesteuert wird, muss die Taschenlampen-App nach einer Berechtigung zum Kamera-Zugriff fragen. Wer diese technischen Zusammenhänge nicht kennt, wird sich wundern und ggf. die Berechtigung verweigern (vgl. Lin et al. 2014, S. 209). Die App kann so nicht zielführend genutzt werden, wird wahrscheinlich wieder gelöscht und womöglich sogar negativ bewertet. Der Hersteller von Android betont in seiner Entwicklerdokumentation zum Thema Berechtigungen dann auch, wie wichtig es ist, den Nutzer*innen gegenüber transparent über den Sinn von Berechtigungen einer App zu sein (online unter developer.android.com).

Welche Beispiele von Software kennen Sie, bei denen aus Nutzer*innen-Sicht keine ausreichende Transparenz im eben geschilderten Sinne vorhanden ist?
Kennen Sie Beispiele für Software, bei denen Transparenz besonders gelungen ist?

Transparente Software als übergreifendes Ideal erscheint gesellschaftlich zunehmend notwendig, zumindest, wenn wir an einer humanistischen Gesellschaft interessiert sind, in der Menschen nicht nur als Konsumenten von Produkten funktionieren. Das Ideal transparenter Software ist die andere Seite der Forderung, nach der auch Nutzer*innen ein zumindest grundlegendes Verständnis von Computertechnik entwickeln müssen, wenn sie weiterhin selbstbestimmt handeln wollen. In seinem Buch „Program or Be Programmed“ bezeichnete Douglas Rushkoff dies als Code Literacy (Rushkoff 2010).

Damit Code Literacy entstehen kann, müssen beide Seiten – Nutzer*innen und Entwickler*innen – an diesem Ziel arbeiten, denn um etwas zu lernen, ist Interesse nötig und die Umgebung muss lernförderlich sein:

- Nutzer*innen haben gewissermaßen eine Holschuld, indem sie bereit sein müssen, grundlegende Funktionsprinzipien von Software verstehen zu lernen. Anstatt mal indifferent, mitunter staunend, oft fluchend vor den Ausgaben eines Programms zu sitzen, sollten sie eine Vorstellung davon entwickeln *wollen*, was das Programm gerade für sie tut, und sich dafür auch mit einfacheren technischen Zusammenhängen auseinandersetzen. Erfreulicherweise gibt es zunehmend Möglichkeiten, algorithmisches Denken zu lernen und praktisch zu üben, ohne dass man in die Tiefen von Programmiersprachen eindringen muss (beispielsweise sogenannte Low-Code-Entwicklungsumgebungen, mit denen man keinen Quelltext schreibt, sondern Programme grafisch modelliert).
- Auf der anderen Seite haben Entwickler*innen eine Bringschuld, indem sie Software so gestalten, dass Nutzer*innen sie verstehen können. Zu transparenter Software gehören im Verständnis vorliegenden Buches beispielsweise:
  - klar verständliche Benutzeroberflächen (im Sinne etwa der schon erwähnten DIN EN ISO 9241-110) …
  - … die die Relevanz ihrer eigenen kommunikativen Ausdrücke sicherstellen und die Relevanz der angestrebten Leistungen der Software herausstellen (vgl. Kap. 3);

– gut kommentierte, einsehbare und leicht auffindbare Quelltexte sowie dazugehörige weitere Projektdateien und Dokumentationen (das bedeutet eine konsequente Entscheidung für Open Source);
– in menschlicher Sprache ausgedrückte Erklärungen, wie Berechnungsergebnisse zustande gekommen sind (dies ist v. a. im Bereich des Machine Learning wichtig).

Das Ideal transparenter Software habe ich in Abb. 1.4 noch einmal zusammengefasst.

Diese Idee von Transparenz ist, wie gesagt, heute nur eine Idealvorstellung. Dem Ideal kann sich nur genähert werden, wenn die Gesellschaft eine andere Einstellung zu Software entwickelt als bisher. Das gilt für Nutzer*innen und Entwickler*innen gleichermaßen:

- Nutzer*innen haben zumindest derzeit kaum Interesse daran, hinter die Oberfläche zu blicken. Die Forderung nach Code Literacy entstand in akademischen Kreisen, die sich ohnehin mit Bildungsarbeit und gesellschaftlicher Entwicklung befassen, aber bei Nutzer*innen herrscht normalerweise ein Gedanke vor: „Es soll einfach nur funktionieren." Man will sich mit Software nicht auseinandersetzen, man will sie nutzen. Man tut so, als wäre Software eine triviale Maschine und verdrängt, dass sie in Wahrheit meistens eine nicht-triviale Maschine ist. Heinz von Foerster sprach von Trivialisierungsbestreben (von Foerster 1993, S. 252). Dass wir damit unsere individuelle Handlungsfähigkeit verlieren und uns abhängig von Dritten machen, sehen viele Menschen gar nicht als ihr persönliches Problem.
- Softwareentwicklung müsste ein Stück weit von kapitalistischen Verwertungslogiken entkoppelt werden. Als Wirtschaftsunternehmen hat man im Allgemeinen kein Interesse daran, dass Nutzer*innen in die Tiefenstruktur blicken können: Nutzer*innen sollen effizient (d. h. vor allem schnell und ohne viel nachdenken zu müssen) ihre angenommenen Nutzungsziele erreichen. Auch mag es Geschäftsgeheimnisse zu schützen geben. Oder man will einfach nicht, dass offenbar wird, dass der eigene Quellcode oder die eigenen Modelle in der Hektik des Arbeitsalltags doch nicht so professionell ausgefallen sind, wie es in der Theorie vorgeschlagen wird.

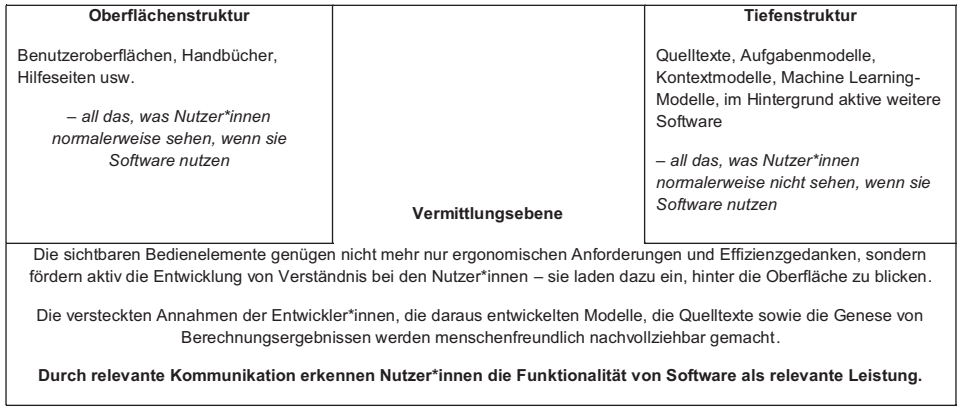

Abb. 1.4 Das Ideal transparenter software

Selbstverständlich kann hier eine Veränderung nur auf freiwilliger Basis geschehen. Wenn man keine Code Literacy entwickeln oder fördern will, wird man für sich selbst ‚gute Gründe' finden. Man ist dann auch kein schlechter Mensch. Aber als Nutzer*in muss man dann unter Umständen mit dem frustrierenden Gefühl leben, keine Kontrolle über die genutzte Technik zu besitzen, weil man in den vorhandenen Machtverhältnissen der unterlegene Teil ist. Und als Entwickler*in müsste man zumindest mögliche kognitive Dissonanzen in moralischer Hinsicht, in Hinblick auf die fortgeschriebenen Machtverhältnisse, bearbeiten.

Vorliegendes Buch jedenfalls soll Ihnen als an Software-Entwicklung beteiligte Person einige nützliche Gedanken und Methoden zur Entwicklung qualitativ gelungener, transparenter Software vermitteln – Software, die Ihre Nutzer*innen ernst nimmt. Wenn Sie bis hierher gelesen haben, wissen Sie schon, dass dafür ein Verständnis menschlicher Kommunikation wichtig ist. Dies schauen wir uns im Folgenden an.

Literatur

Bolchini, Christiana / Curino, Carlo A. / Quintarelli, Elisa / Schreiber, Fabio A. / Tanca, Lettiza: A Data-oriented Survey of Context Models. In: SIGMOD Record, vol. 36, No. 4, December 2007, S. 19–26.

Card, S. / Moran, T. / Newell, A.: The Psychology of Human-Computer Interaction. Hilldale, New Jersey 1983.

Computerwoche (ohne Autor): Programmieren: Kunst oder Handwerk? 07.10.2010. URL: https://www.computerwoche.de/a/programmieren-kunst-oder-handwerk,1230662 (letzter Abruf: 07.11.2019)

Daher, Robil / Donick, Mario / Krohn, Martin / Tavangarian, Djamshid: Quality of Interaction: an Alternative Model for QoS in Interplanetary and Deep-Space Communication Networks. In: Proceedings of 28th AIAA International Communications Satellite Systems Conference (ICSSC-2010), 30.08–02.09.2010, Anaheim/USA, 2010.

Dey, Anind K. / Abowd, Gregory D.: Towards a Better Understanding of Context and Context-Awareness. Technischer Bericht, Georgia Institute of Technology, 1999.

DIN EN ISO 9241-110 Grundsätze der Dialoggestaltung; URL: https://www.iso.org/standard/38009.html (letzter Abruf: 10.09.2019)

Donick, Mario / Daher, Robil / Schwelgengräber, Wiebke / Krohn, Martin / Tavangarian, Djamshid: Messung und Optimierung der Interaktionsgüte (Quality of Interaction) zeitverzögerter CSCW-Anwendungen. In: Praxis der Informationsverarbeitung und Kommunikation, Band 35, Heft 2, 2012, S. 107–112.

Donick, Mario: „Offensichtlich weigert sich Facebook, mir darauf eine Antwort zu geben": Strukturelle Analysen und sinnfunktionale Interpretationen zu Unsicherheit und Ordnung der Computernutzung. Hamburg 2016.

Donick, Mario: Die Unschuld der Maschinen. Technikvertrauen in einer smarten Welt. Wiesbaden 2019.

Dörner, Dietrich (1984): Denken, Problemlösen und Intelligenz. In: Psychologische Rundschau, Band XXXV, Heft 1, 1984, S. 10–20.

von Foerster, Heinz: Wissen und Gewissen. Frankfurt/Main 1993.

Grams, Hans: Die Ekstase des Programmierens. In: Computerwoche, 10.06.1988. URL: https://www.computerwoche.de/a/die-ekstase-des-programmierers,1155445 (letzter Abruf: 07.11.2019)

Hopcraft, John E. / Motwani, Rajeev / Ullmann, Jeffrey D.: Einführung in die Automatentheorie. Formale Sprachen und Komplexitätstheorie. München 2002.

Irrgang, Bernhard: Grundriss der Technikphilosophie. Hermeneutisch-phänomenologische Perspektiven. Würzburg 2009.

ITU 2008: International Telecommunications Union: Recommendation E.800: Definitions of terms related to quality of service; URL: https://www.itu.int/rec/T-REC-E.800-200809-I/en (letzter Abruf: 10.09.2019)

ITU 2001: International Telecommunications Union: Recommendation G.1000: Communications Quality of Service: A framework and definitions; URL: https://www.itu.int/rec/T-REC-G.1000-200111-I/en (letzter Abruf: 10.09.2019)

Kieras, David: A Guide to GOMS Model Usability Evaluation using NGOMSL. 1996. URL: https://www.sciencedirect.com/science/article/pii/B9780444818621500972 (letzter Abruf: 10.09.2019)

Killki, K.: Quality of Experience in Communication Ecosystem. In: Journal of Universal Computer Science, vol. 14, no. 5, 2008, S. 615–624.

Linke, Angelika / Nussbaumer, Markus / Portmann, Paul R.: Studienbuch Linguistik. Tübingen 2004.

Lin, Jialiu / Liu, Bin / Sadeh, Norman / Hong, Jason I.: Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. In: Proceedings of SOUPS 2014, S. 199–212.

Luhmann, Niklas: Die Wissenschaft der Gesellschaft. Frankfurt/Main 1992.

Luhmann, Niklas: Soziale Systeme. Frankfurt/Main 1996.

Luhmann, Niklas: Die Gesellschaft der Gesellschaft. Frankfurt/Main 1998.

Rushkoff, Douglas: Program or Be Programmed. Ten Commands for a Digital Age. New York 2010.

Sneed, Harry: Die meisten Entwickler können nicht programmieren. In: Computerwoche, 31.07.2002. URL: https://www.computerwoche.de/a/die-meisten-entwickler-koennen-nicht-programmieren,531904 (letzter Abruf: 07.11.2019)

Sperber, Dan / Wilson, Deidre: Relevance. Communication and Cognition. Oxford 1995.

Suchman, Lucy: Human-Machine Reconfigurations. Plans and Situated Actions 2nd Edition. Cambridge 2007.

Suhr, Martin: Techne. In: Prechtl, Peter / Burkard, Franz-Peter (Hrsg.): Metzler-Philosophie-Lexikon: Begriffe und Definitionen. Stuttgart 1996, S. 512.

Watzlawick, Paul / Beavin JH / Jackson D: Menschliche Kommunikation: Formen, Störungen, Paradoxien. Bonn 2016.

Willke, Helmut: Technologien des Organisierens und die Krisis des Wissens. In: Gamm, Gerhard / Hetzel, Andreas (Hrsg.): Unbestimmtheitssignaturen der Technik. Eine neue Deutung der technisierten Welt. Bielefeld 2005, S. 127–148.

Zuser, Wolfgang / Biffl, Stefan / Grechenig, Thomas / Köhle, Monika: Software Engineering mit UML und dem Unified Process. München 2001.

Kommunikation bei der Softwareentwicklung

2

Inhaltsverzeichnis

2.1 Kommunikation: Vorausgesetzt, gefordert, aber selten reflektiert

Modelle für die Softwareentwicklung haben sich seit dem alten Wasserfall-Modell (1958) glücklicherweise stark weiterentwickelt. Insbesondere ‚agile Softwareentwicklung‘ (bekannt geworden 2001 mit dem Agilen Manifest, agilemanifesto.org) berücksichtigt den ‚menschlichen Faktor‘, der bei jedem Entwicklungsprozess eine große Rolle spielt. Entsprechend sehen heutige Modelle auch Kommunikation zwischen Menschen als wesentlichen Bestandteil der Entwicklung an. Dies ist auf jeden Fall anzuerkennen und zu fördern. Menschen sind *Menschen*, und Menschen müssen kommunizieren, wenn sie mit einer Tätigkeit erfolgreich sein möchten, die sich an andere Menschen richtet. Schaut man sich aber Lehrbücher, White Papers, Zeitschriftenartikel oder *Wikipedia*-Artikel zu agilen Modellen an, dann fällt aus Sicht der Kommunikationswissenschaft etwas auf, das zu-

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8_2

nächst skeptisch werden lässt: Während Kommunikation zwar vorausgesetzt und gefordert wird, wird doch nur selten reflektiert, was das für die Betroffenen bedeutet.

Um das Problem zu demonstrieren, betrachten wir im Folgenden zwei Beispiele aus Textsorten, die man rezipiert, wenn man sich über ein Entwicklungsmodell das erste Mal informieren möchte: Lexikoneinträge und Lehrbücher. Stellvertretend für andere, schauen wir uns zunächst den *Wikipedia*-Eintrag zu „Extreme Programming" an. Anschließen blicken wir in ein Lehrbuch zur agilen Softwareentwicklung (Rau 2016). Die zu beiden Beispielen gemachten Anmerkungen sind ausdrücklich nicht als Kritik an den Entwicklungsmodellen oder an den Texten zu verstehen – sie sind *Beobachtungen 2. Ordnung* (vgl. Kap. 1), um auf einige blinde Flecken hinzuweisen, die es aus Sicht der Kommunikationspraxis und Kommunikationswissenschaft gibt.

2.1.1 Beispiel 1: *Wikipedia*-Artikel „Extreme Programming"

Da für viele Menschen heute der erste Einstieg in ein Thema oft die *Wikipedia* ist, und wir davon ausgehen können, dass gerade *Wikipedias* technische Fachartikel als Expertenäußerung wahrgenommen werden, beginnen auch wir einmal mit der freien Enzyklopädie. Der *Wikipedia*-Eintrag zum Thema „Extreme Programming" (Stand: 24.08.2019) ist umfangreich und gut strukturiert. Der Artikel erscheint wie eine lesbare Einführung; er hinterlässt nach dem Lesen das Gefühl, gut verstanden zu haben, worum es bei diesem Entwicklungsmodell geht, was seine Besonderheiten und mögliche Schwierigkeiten sind.

Deutlich wird, dass Extreme Programming sehr stark die permanente Einbindung von Kund*innen berücksichtigt, um spontan auf sich verändernde oder sich neu abzeichnende Anforderungen reagieren zu können und frühzeitig Feedback zu erhalten. Das zu entwickelnde Produkt wird als Teamleistung angesehen, es soll daher in einer kommunikationsförderlichen Umgebung gearbeitet werden. Genutzte Verfahren sind unter anderem Paarprogrammierung (das heißt, zwei Programmierer*innen arbeiten gemeinsam am selben Codesegment), die Erstellung, Diskussion, Priorisierung und Verfeinerung kurzer User Storys (um Anwendungsfälle zu notieren, statt ausführliche Use Cases zu analysieren), kurze sogenannte Stand-Up-Meetings, der Austausch zwischen den Programmierer*innen (sogenannte Pair Negotiations) und gegebenenfalls die Erstellung begrifflicher Glossare.

Anforderungen als Geschichte: User Storys

In agilen Entwicklungskonzepten wird ein Verfahren der Anforderungsdefinition verwendet, das großes Potenzial für eine der Nutzungssituation gerechte, kommunikativ reflektierte Entwicklung bietet: die *User Story*. In einer User Story schreiben Ihre Kundin*innen in Alltagssprache ein bis zwei Sätze auf eine Karte, um eine konkrete Anforderung der Nutzer*innen auszudrücken. Die Anforderung soll möglichst kleinteilig und so leicht codierbar sein. Die Sätze können formlos formuliert werden oder einer Vorlage entsprechen, etwa nach dem Schema „Akteur – Handlung – erhoffter

Nutzen“. Zu jeder User Story hält man den geschätzten Zeitaufwand und die Priorität fest (http://www.agilemodeling.com/artifacts/userStory.htm#InitialFormal).

Eine User Story in dieser Form habe ich bei der Entwicklung eines Flugzeugmodells für die Flugsimulation X-Plane aus meiner eigenen Sicht geschrieben (denn ich war sowohl Mitentwickler als auch späterer Nutzer; simuliert wurde das Ultraleichtflugzeug Comco-Ikarus C42 C): „Als Flugschüler möchte ich auch im simulierten Flugzeug bei kaltem Motor den Choke ziehen müssen, damit ich mich an diese Notwendigkeit gewöhne.“ Dabei bezeichnet „Flugschüler“ einen ganz konkreten Akteur (nämlich mich selbst), „bei kaltem Motor den Choke ziehen“ die Handlung und „damit ich mich an diese Notwendigkeit gewöhne“ den erhofften Nutzen.

Eine andere User Story aus diesem Projekt war nicht in Ich-Form formuliert, folgte aber demselben Schema: „Mögliche Interessenten für eine Flugausbildung sollen den gutmütigen und vertrauenswürdigen Charakter der C42 in der Simulation erfahren können, um die Hemmschwelle für einen echten Schnupperflug zu senken.“ Hier bezeichnet „Mögliche Interessenten“ eine anonyme, unbekannte Menge von Akteuren, „den Charakter des Flugzeugs erfahren“ die Handlung und „um die Hemmschwelle für einen Schnupperflug zu senken“ den Nutzen. Die Anforderung ist weniger konkret als im ersten Beispiel: um einen spezifischen ‚Charakter‘ eines Flugzeugs zu simulieren, reicht es nicht, die ‚harten‘ Daten des zu simulierenden Modells umzusetzen, sondern man muss sich dem ‚richtigen‘ Verhalten in zahlreichen Iterationen annähern.

Erst während des Tests der ersten Versionen unseres Produkts kamen weitere Anforderungen hinzu. Da wir User Storys nicht konsequent verwendet haben, wurden die neuen Anforderungen nur noch informell notiert, beispielsweise: „Die Propellerdrehzahl sollte für die ersten sieben bis acht Sekunden nach dem Motorstart runtergeregelt sein“. Hier fehlen Akteur und Nutzen. Als formale User Story könnte die Anforderung wie folgt lauten: „Als Flugschüler möchte ich, dass die Propellerdrehzahl kurz nach dem Start auf maximal 1700 Umdrehungen runtergeregelt ist, damit ich einen realistischen Eindruck vom Motorstart erhalte.“ Hier sind Akteur und Nutzen ergänzt, und die Anforderung ist direkt codierbar.

Im Folgenden habe ich fünf, mir im Kontext vorliegenden Buches besonders prägnant erscheinende, Aussagen aus dem *Wikipedia*-Artikel „Extreme Programming“ zitiert. Sie zeigen, dass das Problem der Kommunikation auf jeden Fall erkannt wurde:

1. „Der allgemeine Wissensaustausch und die stetige Kommunikation beugen einem Wissensmonopol vor.“
2. „Ein ehrlicher Umgang mit dem Kunden soll die Glaubwürdigkeit und Zufriedenheit steigern und die Angst minimieren, die unter Umständen zwischen Kunde (‚Haben die mich verstanden? Was werden die wohl liefern?‘) und Entwicklung (‚Was will er eigentlich genau?‘, ‚Ob er zufrieden sein wird mit dem, was wir liefern?‘) vorherrscht.“
3. „Aufwandsabschätzungen werden verlässlicher, da sie im Team getroffen […] werden.“

4. „In XP soll nur das verwirklicht werden, was tatsächlich einen Nutzen für den Kunden hat."
5. „Das meiste Wissen über die Funktionalität ihrer Entwicklung [der User Storys] befindet sich in den Köpfen der Beteiligten."

Obwohl die zitierten Aussagen alle in eine unbedingt zu unterstützende Richtung zielen, sind aus Sicht der Kommunikationspraxis und der Kommunikationswissenschaft doch einige Anmerkungen zu machen bzw. Fragen zu stellen (1a bis 5a):

1a. Was ist mit „Wissen" gemeint? Ist das der eher diffuse Alltagsbegriff von Wissen? Oder ein elaborierter Begriff von *Wissen*, der das komplexe kognitive Gefüge aus aktiven und trägen Elementen benennt, das nicht nur bewusst abrufbares Sachwissen (Daten, Fakten usw.), sondern auch teils unbewusst vorhandene Kompetenzen umfassen kann? Oder meint „Wissen" vor allem projekt- und produktrelevante *Daten*, die gegebenenfalls als *Informationen* kognitiv repräsentiert und als relevant erkannt wurden (vgl. Sperber und Wilson 1995)? Dies zu klären, ist eigentlich nötig, wenn man Form und Häufigkeit der „stetig[en] Kommunikation" bestimmen will.

2a. Das ist ein sehr wesentlicher Aspekt. Er geht aber nicht weit genug. In der zitierten Form stellt der Aspekt nur sicher, dass die Personen, die die jeweilige Kundeninstanz formal repräsentieren (in Meetings usw.), ausreichend berücksichtigt sind. Diese Personen sind aber oft gar nicht selbst repräsentativ für die Mehrheit tatsächlicher Benutzer*innen. Wenn es überhaupt selbst Nutzer*innen sind, dann oft welche, die sich schon ein gewisses Expertentum erworben haben (in der agilen Methode „Crystal" werden sie als „kundige Benutzer" bezeichnet). Damit unterscheidet sich deren Sicht aber von weniger involvierten Nutzer*innen, die daher ebenfalls einbezogen werden sollten (vgl. Kasten „Einbeziehung unkundiger Nutzer*innen").

3a. Es gibt einen alten Scherz, der die Skepsis gegenüber Teamarbeit ausdrückt: „**T**oll, **e**in **a**nderer **m**acht's". Dies drückt aus, dass man sich in einem Team auch verstecken kann – man erbringt weniger Leistung als möglich, weil die Teamkolleg*innen das schon rausreißen werden. Oder man fühlt sich als Teammitglied weniger wertgeschätzt, als wäre man individuell für einen eigenen Aufgabenbereich verantwortlich. Skepsis gegenüber Teams kann die Kommunikation im Team beeinträchtigen. Vorteile wie der in Punkt 3 genannte können dann nicht zum Tragen kommen. Es ist also zu fragen, wie innerhalb eines Teams überhaupt erfolgreiche Kommunikation sichergestellt werden soll. Wie wird der Aufbau des Teams betrieben, wie wird Konfliktmanagement gestaltet?

4a. Dass nur das entwickelt werden soll, was wirklich einen Nutzen hat, ist ein Ziel, dem man nur zustimmen kann. Da der tatsächliche Nutzen einer Software sich aber erst in der konkreten Nutzungssituation zeigt, sind auch die mit viel Kommunikation einhergehenden, iterativen Anforderungsanalysen und -anpassungen nur eine Annäherung. Der tatsächliche Nutzen kann erst durch Beobachtung tatsächlicher Nutzer*innen in tatsächlichen Nutzungssituationen beurteilt werden. Hier unterscheiden sich Extreme Programming und andere agile Methoden nicht vom alten Wasserfall-Modell.

5a. Diese Aussage, die mit Aussage 1 gewissermaßen einen Rahmen bildet, drückt den Kern des ganzen Problems von Entwicklung aus. Ob nun in knapper Form User Storys erstellt und priorisiert, oder detailliertere Use Cases entwickelt werden – es sind alles nur Dokumente, die erst vor dem Hintergrund subjektiver Kontexte relevant werden. Extreme Programming will dies durch Kommunikation erreichen. Aber es muss sichergestellt werden, dass diese Kommunikation tatsächlich zu Verständigung führt und nicht etwa von Missverständnissen aufgrund unterschiedlicher Annahmen getrübt ist.

Die Anmerkungen sind, wie gesagt, weder als Kritik am zitierten *Wikipedia*-Artikel noch an Extreme Programming zu verstehen. Sie weisen lediglich auf blinde Flecken hin, die sich so auch in anderen Entwicklungsmodellen finden.

Einbeziehung ‚unkundiger' Nutzer*innen

Denken Sie immer daran, dass Ihr Produkt am Ende auch von *unkundigen Nutzer*innen* verwendet wird (den abwertenden Begriff des DAU, also des „Dümmsten Anzunehmenden Users" halte ich nicht für angemessen).

Diese Menschen mögen im Job zum Einsatz Ihres Produkts gezwungen sein (vielleicht werden sie sogar von der Person dazu gezwungen, die Ihre Kontaktperson bei Ihrem Kunden ist), oder sie mögen sich freiwillig für Ihr Produkt entschieden haben, überfordern sich damit aber. Ich schreibe hier von den wirklich vielen Menschen, die sich überhaupt nur schwer in einer Benutzeroberfläche zurechtfinden. Personen, die nie das Prinzip von Eingabefeldern, Links, Scrollbalken, Menüs usw. verstanden haben, sondern stattdessen mühsam Klickwege auswendig lernen und hilflos sind, wenn sich nur der Name eines Menüpunkts ändert. Diese Menschen können von vermeintlich effizienteren Neuerungen so verunsichert werden, dass Wochen der Eingewöhnung nötig sind.

Entwickeln Sie nun beispielsweise für ein großes Unternehmen das Web-Interface einer CRM-Anwendung, die von vielen hundert Sachbearbeiter*innen benutzt wird, dann beziehen Sie bitte auch die unkundigeren unter diesen Mitarbeiter*innen in die Entwicklung ein. Entwickeln Sie eine Anwendung für eine anonyme Menge potenzieller Nutzer*innen, deren Kenntnisse und Fähigkeiten sehr heterogen sein könnten, holen Sie sich bewusst unkundige Testnutzer*innen. Gehen Sie davon aus, dass scheinbare Selbstverständlichkeiten oft keine sind. Führen Sie Beobachtungen durch, um Fallstricke zu erkennen. Die Verfahren in diesem Buch (vgl. Kap. 4, 5 und 6) bieten Ihnen eine geeignete Möglichkeit, entsprechende Beobachtungen strukturiert und effizient durchzuführen.

2.1.2 Beispiel 2: Ein Lehrbuch zur Agilen Softwareentwicklung

Stellvertretend für andere Lehr- und Fachbücher schauen wir in das recht aktuelle Lehrbuch „Agile objektorientierte Software-Entwicklung" (Rau 2016). Das Buch ist eine ausführliche Einführung. An einem vollständigen Fallbeispiel erklärt es die *Java*-basierte

Entwicklung einer Software-Lösung mit agilen Methoden. Da das Buch selbst grundlegende Begriffe einführt und das Fallbeispiel mit zahlreichen Quelltexten demonstriert, eignet es sich sowohl für Studierende an einer Hochschule als auch für das Selbststudium.

An zahlreichen Stellen des Bandes wird die Wichtigkeit von Kommunikation für agile Entwicklungsprozesse betont. Fünf besonders prägnante Aussagen möchte ich im Folgenden zunächst zitieren und danach kommentieren:

1. „die User Story [ist] eine Art Notiz, die dazu geeignet ist, Analysegespräche einzuleiten“ (ebd., S. 45). „Werden im Gespräch mit dem Anwender mehrere Qualitätseigenschaften […] genannt, so kann über die Frage ‚Werden wirklich alle sofort benötigt?‘ eine Priorisierung identifiziert werden und unterschiedliche User Stories formuliert werden“ (ebd., S. 46).
2. „[…] sollte sichergestellt werden, dass alle Beteiligten die gleiche Sprache sprechen“ (ebd., S. 49).
3. „Im weiteren Entwicklungsprozess dient das Glossar einer eindeutigen Verständigung, unterliegt einer kontinuierlichen Entwicklung und kann […] zur Erstellung von Hilfetexten verwendet werden“ (ebd., S. 50).
4. „die Demonstration der potenziell einsetzbaren Software […] ermöglicht die direkte Kommunikation zwischen Teammitgliedern und den Benutzern“ (ebd., S. 247).
5. „Daher haben sich das Entwicklerteam und der Produktverantwortliche dafür entschieden, die Software-Lösung nicht nur zu präsentieren, sondern [betroffenen Nutzer*innen] die Möglichkeit zu geben, [die Software testweise zu nutzen]“ (ebd.).

Wir sehen an den zitierten Textstellen, dass auch in der informatischen Lehre die große Bedeutung menschlicher Kommunikation erkannt wurde. Dies entspricht auch den Erfahrungen, die im Arbeitsalltag in interdisziplinären Projekten gesammelt werden können. Allerdings sind aus Sicht der Kommunikationspraxis und -wissenschaft wiederum einige Anmerkungen zu machen oder Fragen zu stellen:

1a. Die Idee, User Storys wie eine Art Gesprächsleitfaden zu verwenden, ist nachvollziehbar. Es ist auch zu erwarten, dass sich diese in Gesprächen auf bestimmte Eigenschaften konkretisieren. Die Herausforderung besteht darin, aus diesen Gesprächen ‚das Richtige‘ mitzunehmen, das heißt sicherzustellen, dass die Anwender*innen unter diesen Eigenschaften möglichst dasselbe verstehen wie die Entwickler*innen.

2a. Dies wird auch von Rau (2016) erkannt. Freilich gehört zum Sprechen derselben Sprache mehr als nur das Teilen des gleichen Codes. Auch die Hintergründe müssen auf einen vergleichbaren Stand gebracht werden, damit nicht unter derselben sprachlichen Form doch etwas anderes verstanden wird.

3a. Ein dazu (nicht nur in der Softwareentwicklung, sondern auch in wissenschaftlichen interdisziplinären Zusammenhängen) häufig verfolgter Weg ist das Anlegen eines Glossars; darin werden benötigte Begriffe so definiert, dass alle Beteiligten der Definition zustimmen und bei Bedarf nachschlagen können. In der Variante Extreme Pro-

gramming war früher auch die Formulierung von Metaphern aus dem Alltagsbereich vorgesehen, um zwischen unterschiedlichen Verständigungshintergründen zu vermitteln. Sowohl Glossar als auch Metaphern lösen das Problem unterschiedlichen Verstehens nur teilweise – denn sie selbst können (und werden) unterschiedlich verstanden.

4a. Dieser Punkt ist hervorzuheben. Die frühzeitige und häufige Einbeziehung von Nutzer*innen ist, wie auch im Extreme Programming, der große positive Aspekt agiler Methoden. Eine bloße Demonstration mit Fragen und Diskussion nutzt das Potenzial dieses Zusammentreffens aber nicht unbedingt aus.

5a. Wie auch Rau (2016) erkennt: Er berichtet daher in dem Zitat von einer Software, die im Hochschulkontext für die Verwaltung von Studierendenaustauschen entwickelt wurde. Die Betroffenen selbst, d. h. Studierende, die ein Auslandssemester absolvieren wollten, wurden gebeten, Funktionalitäten der Software auszuprobieren. In Diskussionen ergaben sich dann Änderungen an den vorab entwickelten User Stories, was die weitere Entwicklung positiv beeinflusste.

2.1.3 Was uns die Beispiele verdeutlichen

Die beiden Beispiele (*Wikipedia*-Artikel einerseits, Lehrbuch andererseits) ließen sich durch zahlreiche weitere Texte ähnlicher Art ergänzen. Sie zeigen,

- dass Kommunikation in der Software-Entwicklung vorausgesetzt und gefordert wird, dass aber selten auf Faktoren eingegangen wird, die den Erfolg oder Misserfolg kommunikativer Verständigung bedingen;
- dass die Unwahrscheinlichkeit der Kommunikation (Luhmann) zwar erkannt, aber nicht bewusst reflektiert wird (oder höchstens insoweit, als Oberflächenphänomene bearbeitbar erscheinen, etwa durch die Erstellung eines Glossars, um Begriffe scheinbar eindeutig zu definieren);
- dass die Rolle von Nutzer*innen für die Gestaltung von Software erkannt wird, aber vor allem Gespräche mit Nutzer*innen dominieren, während der Wert strukturierter Beobachtungen von Nutzer*innen für die Identifikation von Anwendungsfällen und für die Prüfung der Angemessenheit des Produkts für die identifizierten Anwendungsfälle eher zweitrangig erscheint.

Zusammenfassend halten wir daher fest:

> Die Leistung von Software ist eine Beobachtung ihrer Nutzer*innen. Darum zeigt sie sich nicht während der Entwicklung, sondern erst in der tatsächlichen Nutzungssituation. Es ist daher notwendig, ein umfassendes Verständnis dieser Nutzungssituationen zu erlangen. Zum Erlangen dieses Verständnisses ist *Kommunikation über potenzielle Nutzungssituationen* ein wichtiger Schritt. Für einen Abgleich der kommunizierten Erwartungen mit tatsächlich stattfindender Nutzung muss jedoch die *Beobachtung tatsächlicher Nutzungssituationen* einbezogen werden.

Nutzer*innen beim Umgang mit Software zu beobachten und sie dabei etwa ‚laut denken' zu lassen (sie also ihre spontanen Gedanken und Eindrücke ausdrücken zu lassen; vgl. Someron et al. 1994), führt zu anderen Ergebnissen als sich mit Nutzer*innen zu unterhalten. In der Kommunikation untereinander achten Menschen viel eher darauf, welchen Eindruck sie beim Gegenüber erzeugen (Impression Management), als wenn sie in einer quasi-natürlichen Nutzungssituation vor sich her sprechen (es ist erwiesen, dass Nutzer*innen beim ‚lauten Denken' schnell so sprechen, wie sie es auch unbeobachtet tun würden, vgl. ebd., S. 26). Darin liegt ein großes Potenzial.

Wie Sie solche, am ‚lauten Denken' orientierte Beobachtungen auf strukturierte Weise durchführen können, um so das von Ihnen präferierte Entwicklungsmodell zu ergänzen, lernen Sie in vorliegendem Buch (vgl. Kap. 3, 4, 5 und 6). Um dafür aber eine Grundlage zu haben, schauen wir uns vorher an, was das Besondere an menschlicher Kommunikation ist.

2.2 Modelle menschlicher Kommunikation

2.2.1 Kommunikation als Übertragung vom Sender zum Empfänger

Über Kommunikation machen wir uns meistens wenig Gedanken. Wir gehen davon aus, dass wir es können. Solange wir dieselbe Sprache sprechen und vergleichbare Hintergründe haben (wie etwa Alter, Geschlecht, Kultur, Ausbildung oder Beruf), werden wir uns schon verstehen. Und haben wir doch einmal den Eindruck, dass wir missverstanden wurden, sprechen wir das kurz an – gerne in der Form „Da haben *Sie* mich missverstanden", die das Missverstehen der anderen Person zuschreibt, vielleicht aber auch in der selbstreflexiven Form „Da habe ich mich wohl unklar ausgedrückt." Beide Seiten verweisen auf ein traditionelles Modell der Kommunikation, das ursprünglich als mathematisches Modell in der Nachrichtentechnik entstand, aber lange Zeit auch die Arbeiten zur menschlichen Kommunikation beeinflusste: das Sender-Empfänger-Modell von Claude Shannon (Shannon 1948, vgl. Abb. 2.1).

Ausgehend von einer Informationsquelle werden durch einen Sender Signale generiert, die über einen Kanal zu einem Empfänger und dort in die Informationssenke gelangen.

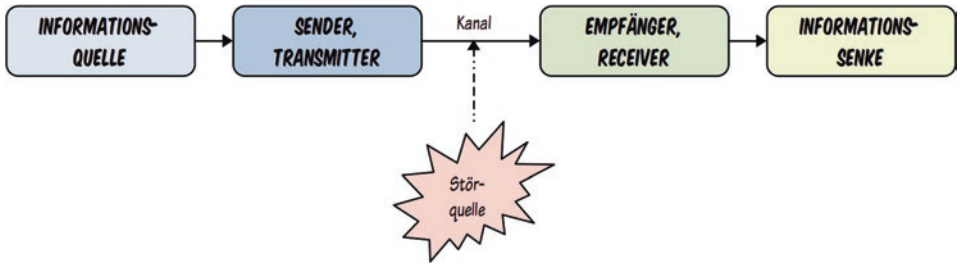

Abb. 2.1 Traditionelles Sender-Empfänger-Modell der Kommunikation (eigene Darstellung)

Unter Umständen wird das Signal auf seinem Weg durch Störsignale von einer Störquelle gestört, sodass es nicht in originaler Form beim Empfänger ankommt.

Dieses Modell stellt den Übertragungsweg und das Signal einer Nachricht in den Vordergrund, und so betrachtete man auch menschliche Kommunikation lange als relativ einfachen Übertragungsvorgang. Daran ist zunächst auch nichts verkehrt, denn vor allem technische Probleme der Kommunikation können so bereits berücksichtigt werden:

- Jack stellt Jill eine Frage. Er formuliert die Frage in Gedanken (Informationsquelle), bevor sein Mund (Sender) die Wörter formt, die als Schallwellen (Signal) durch die Luft (Kanal) an Jills Ohr (Empfänger) dringen, die von ihrem Gehirn verarbeitet werden und wieder zu Gedanken werden (Informationssenke). Vielleicht ist es laut im Büro (Störsignal), weil ein anderer Kollege telefoniert (Störquelle), dann kommen die Schallwellen verzerrt bei Jill an und sie versteht nicht alles. Vielleicht spricht Jack aber auch eine andere Sprache, und weil Jill Jacks Code nicht decodieren kann, versteht sie ihn nicht (der Begriff Informationssenke wäre daher hier verfehlt, denn damit ein Datum zu einer Information wird, muss man es als relevant für den eigenen Kontext erkennen, also verstehen).
- Max ist ein neuer Programmierer im Team und arbeitet sich in Quellcode (Signal/Kanal) ein, den er von seinem Vorgänger Moritz (Sender) ‚geerbt' hat. Moritz hat seinen Code gut dokumentiert, damit seine Gedankengänge nachvollziehbar sind (Informationsquelle). Max (Empfänger) kommt gut mit dem Code zurecht und versteht ihn schnell (Informationssenke).
- Anna (Sender) schreibt ein paar User Storys. Sie notiert ihre Gedanken (Informationsquelle) mit einem Kugelschreiber auf Karteikarten (Signal/Kanal), die sie am Ende auf Hannahs Schreibtisch legt. Aus Unachtsamkeit kippt Hannah ihre Kaffeetasse um (Störquelle), sodass auf einigen Karten die Schrift verwischt und unleserlich wird (Störsignal). Nachdem sie ihren Schreibtisch in Ordnung gebracht hat, liest Hannah (Empfänger) die noch entzifferbaren User Storys, kann damit aber nicht viel anfangen – neben der verwischten Schrift hat Anna auch Fachbegriffe aus ihrem Arbeitsumfeld verwendet, die Hannah nicht kennt (wieder wäre es verfehlt, von Informationssenke zu sprechen).

Wie Sie sehen, kann man zwar verschiedene Kommunikationssituationen in Form des Sender-Empfänger-Modells abbilden, und wenn es, wie im zweiten Beispiel, keine Probleme gibt, reicht das auch aus. Aber im ersten und dritten Beispiel bleiben doch Fragen. Es gab dort zwar jeweils auch Probleme auf dem Übertragungsweg, die durch Störquelle und Störsignal bedingt sind, aber für die Verständigung war das nebensächlich. Wichtiger war, dass Jill die Sprache von Jack nicht gesprochen hat, und dass Hannah mit Annas Fachwortschatz nicht zurechtgekommen ist. Diese Aspekte liegen außerhalb der Reichweite des Sender-Empfänger-Modells.

Deshalb wurde in der Kommunikationsforschung bald erkannt, dass das Sender-Empfänger-Modell in seiner reinen Form nicht ausreichend ist. Man musste irgendwo berücksichtigen, dass Menschen unterschiedliche Verstehenshintergründe haben, die bei gleichem Code eine

Verständigung erschweren oder erleichtern können. Manchmal nennt man dies „Weltwissen“ (Linke et al. 2004, S. 257), manchmal spricht man vom „common ground“ (Clark 1996). Entsprechend modifizierte Modelle stellen diese Hintergründe mitunter als Schnittmenge zwischen Sender und Empfänger dar. Noch eine Erweiterung betrifft die Reziprozität (Gegenseitigkeit) der Kommunikation – ein Empfänger wird selbst zum Sender, wenn er eine Antwort schickt. Hier haben sich die Modelle meist mit einem Pfeil in Gegenrichtung beholfen. Diese erweiterten Modelle kommen der Komplexität menschlicher Kommunikation schon etwas näher, denn sie lenken das Augenmerk weg vom Übertragungsvorgang und hin zu Problemen der Verständigung. Sie tun allerdings nicht mehr als das. Wie es zu Verständigung kommt – und dann noch zu erfolgreicher – können solche Modelle nicht erklären.

Der Arbeitswissenschaftler Thomas Herrmann hat ein Modell vorgeschlagen, das diesen Mangel zu beheben versucht (Herrmann 2001). Insbesondere präzisiert Herrmann die Verstehenshintergründe der beteiligten Personen, indem er von einem inneren und äußeren Kontext spricht. Beides beeinflusst die Produktion und Rezeption kommunikativer Akte; Herrmann spricht von Ausdruckserzeugung auf Senderseite und Eindruckserzeugung auf Empfängerseite. Der innere Kontext umfasst etwa das Wissen, das man über ein Thema bereits besitzt, aber auch Vorstellungen, Wünsche, Erwartungen, sowie Partnerbilder (ebd., S. 17). Der äußere Kontext beschreibt weitere Parameter der Kommunikationssituation, wie Ort, Zeit oder soziale Gegebenheiten. Herrmann zählt auch die Geschichte der bisher ausgetauschten Ausdrücke dazu (ebd., S. 18). Teilweise sind diese Parameter durch alle beteiligten Personen wahrnehmbar, teilweise nur durch einzelne. So entstehen natürlich unterschiedliche Perspektiven auf die Situation.

In Herrmanns Modell wird deutlich, dass für Verständigung nicht nur Code, Signal und ‚Weltwissen‘ nötig sind (letzteres wäre Teil des inneren Kontexts), sondern dass Kommunikation von noch ganz anderen Parametern abhängen kann. Gleichwohl liegt dem Modell immer noch eine Paketmetapher zugrunde: Ein Sender produziert etwas, verpackt es, verschickt es, und dann muss der Empfänger sehen, was er daraus macht – ein klar definierter Prozess, so als würden wir technische Datenpakete verschicken. Zudem vereinfacht Herrmann ein wesentliches Problem erfolgreicher Verständigung, indem er davon ausgeht, dass wir wissen können, was eine andere Person bereits weiß und es daher nicht extra ansprechen müssen. Herrmann schreibt: „A(ndrea) braucht all das nicht explizit auszudrücken, was B(ert) aus eigener Wahrnehmung weiß, also das, was bereits zu seinem *inneren Kontext* […] gehört“ (ebd., S. 17; Hervor. i.O.). Es stimmt, dass wir im Alltag oft so verfahren, aber gerade das führt eben zu Missverständnissen – die andere Person weiß vielleicht doch etwas *anderes* oder weiß etwas *anders* als ich es erwartet habe, und ich selbst habe letztlich nur Annahmen über das Wissen der anderen Person. Diesen Vagheiten menschlicher Kommunikation werden Sender-Empfänger-Modelle auch in so elaborierten Formen wie bei Herrmann nicht gerecht.

2.2.2 Kommunikation als Beziehung und Haltung

Schon in den 1930er Jahren hat Karl Bühler (1879–1963) sein Organon-Modell vorgestellt. Bühler bezieht sich auf den antiken griechischen Philosophen Platon, der Sprache als *organum*, also als Werkzeug, bezeichnet hat, „um einer dem andern etwas mitzuteilen

über die Dinge“ (Bühler 1965, S. 24). Auf diesen drei „Relationsfundamente[n]“ (ebd., S. 25), also „einer“, „dem andern“ und „die Dinge“, baut Bühler ein komplexes Modell auf, dass die Beziehung von Sender, Empfänger, Sache und „Schallphänomen“ (ebd.) beschreibt (Abb. 2.2).

Das Schallphänomen wird in dreifacher Hinsicht zum Zeichen: Es ist Symbol, denn es stellt einen Gegenstand oder Sachverhalt dar (ebd.); es ist „Symptom“ (ebd.), denn es drückt die „Innerlichkeit“ (ebd.) des Senders aus; und es ist Signal, denn es richtet einen Appell an den Empfänger (ebd.) – den man übrigens am besten anhand der Beobachtung des Empfängers und seiner Reaktionen erkenne (ebd., S. 31). Bühler spricht hinsichtlich des Zeichencharakters der Schallphänomene auch von semantischen Funktionen (ebd.) oder „Sinnfunktionen“ (ebd., S. 32), weil sie erklären, wie das Schallphänomen sinnhaft werden kann. Der Begriff der Sinnfunktion wird uns noch wiederbegegnen, wenn wir auf die Formen der Kommunikation zu sprechen kommen, die der Luhmann-Schüler Dirk Baecker vorgeschlagen hat (Baecker 2007) und die ich in Donick (2016) empirisch genutzt habe (vgl. Abschn. 2.2.3 und Kap. 3). Statt Schallphänomenen kann man natürlich auch andere Phänomene als materiale Grundlage von Zeichen nehmen, zum Beispiel Schrift oder grafische Darstellungen.

Der russische Philologe Roman Jakobson (1896–1982) hat, teils in Erweiterung Bühlers, eine Theorie der Sprachfunktionen entwickelt (Tab. 2.1). Jakobson unterscheidet sechs Funktionen der Sprache, die von sechs Faktoren bestimmt werden. Die Faktoren kennen wir bereits: Es sind Sender (addresser), Empfänger (addressee), Nachricht (message), Kontext (context), Kontakt (contact) und Code (Jakobson 1960, S. 3). Diese Faktoren als solche bilden in Jakobsons Darstellung zunächst ein gewöhnliches Sender-Empfänger-Modell, wie wir es seit Bühler und Shannon kennen. Dies ist aber nur die Basis für Jakobsons Diskussion der Funktionen, die Sprache in der Kommunikation ein-

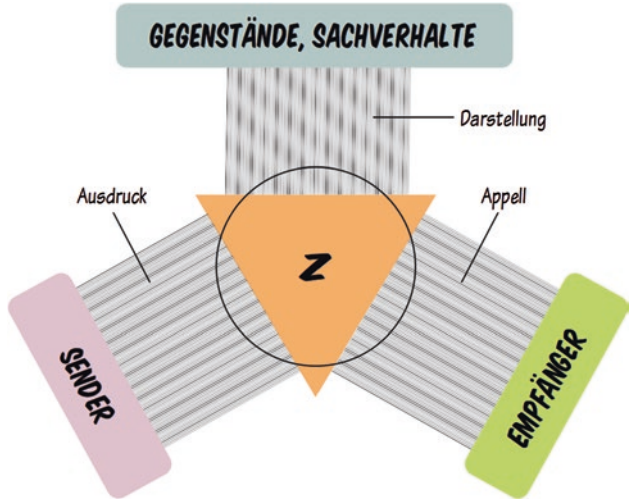

Abb. 2.2 Karl Bühlers Organon-Modell (eigene Darstellung)

Tab. 2.1 Sprachfunktionen nach Roman Jakobson (1960)

Faktor	Funktion	Erklärung
context (Kontext)	referential (referenziell)	die Orientierung an einem Kontext ist ein wesentlicher Aspekt für die meisten Nachrichten (Jakobson 1960, S. 4)
addresser (Sender)	emotive (emotiv)	der Ausdruck der Einstellungen, die der Sender zum Gesprochenen hat; oft mit dem Ausdruck von Emotion verbunden (ebd.)
addressee (Empfänger)	conative (konativ)	die konative (= strebende, begehrende) Einstellung zum Empfänger (ebd.), d. h. der Appell an den Empfänger
contact (Kontakt)	phatic (phatisch)	die Etablierung, Aufrechterhaltung oder Beendigung der Kommunikation; das Prüfen der Funktionstüchtigkeit des Kanals; das Wecken der Aufmerksamkeit des Empfängers (ebd., S. 5)
code (Code)	metalingual (metasprachlich)	die Prüfung, ob Sender und Empfänger denselben Code verwenden, z. B. dieselbe Sprache sprechen (ebd., S. 5f.)
message (Nachricht)	poetic (poetisch)	Einstellungen zur Form der Nachricht als solche; nicht beschränkt auf Poesie als Kunstform; ein Beispiel sind Sprechweisen, die man wählt, weil sie angenehmer klingen als andere (ebd., S. 6).

nimmt. Die Zuordnung der Funktionen ist in Tab. 2.1 in der Reihenfolge ihrer Einführung durch Jakobson gezeigt. Auffällig ist, dass Jakobson den Kontext als erstes diskutiert und erst zum Schluss auf die Nachricht als solche zu sprechen kommt. Hier wird schon eine andere Gewichtung deutlich.

Die ersten drei Faktoren und Funktionen (context & referential; addresser & emotive; addressee & conative) haben wir so fast identisch bei Bühler gesehen. Die weiteren drei Faktoren und Funktionen (contact & phatic; code & metalingual; message & poetic) hat Jakobson in die Diskussion eingebracht. Bei der Produktion und Rezeption einer Nachricht wirken alle sechs Funktionen zusammen, sind für das Verständnis der Nachricht von Bedeutung und können bei Bedarf selbst zum Thema gemacht werden. Dabei herrscht keine Eindeutigkeit.

Beispielsweise führe ich in diesem Buch öfter fiktionale Sprecher*innen-Paare an, wie Jack und Jill – und das tue ich in einer bestimmten Reihenfolge. Mit Jakobsons Sprachfunktionen könnte man nun fragen, warum ich immer denselben Namen zuerst nenne und den anderen zuletzt (Jakobson selbst gibt ein ähnliches Beispiel, vgl. ebd., S. 6). Liegt das an der referenziellen Sprachfunktion, also dem Kontext? Bevorzuge ich zum Beispiel die Person Jack gegenüber der Person Jill? Dies wäre ein sozialer Kontext. Oder halte ich mich nur an die übliche Konvention, dass mindestens seit dem traditionellen englischen Kinderreim („Jack and Jill went up the hill“, 18. Jahrhundert) Jack immer vor Jill genannt wird, und seitdem auch in vielen Büchern, die Jack und Jill als Beispiel benutzen? Diese Konvention wäre ein kultureller Kontext, vor dem mein Handeln Sinn ergeben würde. Aber vielleicht gefällt mir auch einfach der Klang besser – Jack und Jill geht mir ‚flüssiger‘ von den Lippen als es Jill und Jack täten; hier stünde nicht die referenzielle Sprach-

funktion, sondern die poetische Sprachfunktion im Zentrum. (*Dass* mir diese Reihenfolge besser gefällt, könnte wiederum am kulturellen Kontext liegen, den ich vielleicht so verinnerlicht habe, dass er mir wie selbstverständlich erscheint). Sprachfunktionen geben uns wichtige Hinweise an die Hand, warum wir auf bestimmte Weise kommunizieren, aber sie sind genauso wenig eindeutig, wie es Kontexte sind.

Strukturalismus, Poststrukturalismus und „Die siebte Sprachfunktion"

Roman Jakobson gehört zu den sogenannten Strukturalisten, also einer sprachphilosophischen Richtung, die davon ausgeht, dass Sprache symbolhaft auf Wirklichkeit verweist, die Sprache anhand ihrer Strukturmerkmale erklärt und scheinbar allgemeingültige Regelmäßigkeiten ‚entdeckt' (Jakobsons Sprachfunktionen sind ein Beispiel dafür). Bekannte Strukturalisten neben Jakobson waren Ferdinand de Saussure (1857–1913), Claude Lévi-Strauss (1908–2009), Jacques Lacan (1901–1981) und Gilles Deleuze (1925–1995). Ihnen entgegen stellten sich seit den 1960er Jahren Poststrukturalisten wie Michel Foucault (1926–1984), Jacques Derrida (1930–2004), Roland Barthes (1915–1980), Jean Baudrillard (1929–2007), Louis Althusser (1918–1990), Julia Kristeva (*1941), Judith Butler (*1956) und Slavoj Žižek (*1949). Poststrukturalisten gehen davon aus, dass Sprache nicht einfach auf Wirklichkeit verweist, sondern dass wir mit dem Gebrauch von Sprache erst die Unterscheidungen treffen, mit denen wir unsere soziale Wirklichkeit konstruieren (vgl. Abschn. 2.2.3).

Es macht beispielsweise einen Unterschied, ob ich in einem Buch das generische Maskulinum verwende („Liebe Leser!") oder ob ich versuche, die Diversität von Geschlechterkonstruktionen zu berücksichtigen („Liebe Leser*innen!") – meine Entscheidung für eine sprachliche Form ‚baut' an der sozialen Wirklichkeit mit, die ich mir wünsche oder die ich unbewusst unterstelle. Daher hat der Poststrukturalismus zahlreiche Bezugspunkte zum Konstruktivismus, einer erkenntnistheoretischen Position, nach der wir nichts über eine möglicherweise vorhandene ‚objektive' Wirklichkeit sagen, sondern nur individuelle Konstruktionen entwickeln können.

Im Roman „Die siebte Sprachfunktion" (2016) geht der französische Schriftsteller Laurent Benier von der Idee aus, dass Roman Jakobson eine geheime, siebte Sprachfunktion entdeckt hätte, die die menschliche Kommunikation für immer verändern könnte. Das kommt dem französischen Präsidentschaftskandidaten Francois Mitterand Anfang der 1980er Jahre gerade recht. Und so setzen Politik und konkurrierende Forscher*innen alles daran, einen Aufsatz Jakobsons über diese mysteriöse Funktion in die Hände zu bekommen. In der Form eines Kriminalromans, der seine eigene Form zunehmend poststrukturalistisch reflektiert, stellt Benier das akademische Milieu dar, in dem Strukturalisten und Poststrukturalisten lebten, arbeiteten und aufeinandertrafen. Das Buch ist nicht nur wegen der spannenden Geschichte empfehlenswert, sondern auch wegen der teils skurrilen und sehr komischen Situationen mit den sehr exzentrisch gezeichneten berühmten Personen.

In der Tradition von Bühlers Organon-Modell und Jakobsons Sprachfunktionen hat der Psychologe Friedemann Schulz von Thun einen sehr bekannten Ansatz zur Beschreibung menschlicher Kommunikation entwickelt (vgl. Schulz v. Thun 2010). Er ist bekannt als 4-Ohren-Modell oder 4-Seiten-Modell der Kommunikation (Abb. 2.3).

Das Modell fokussiert sich auf die Eindrücke, die eine Nachricht beim Empfänger erzeugen kann. Friedemann Schulz von Thun verwendet das Modell, um die vielfältigen Verständigungsprobleme zu klären, die es zwischen Menschen geben kann. Das Modell geht davon aus, dass eine Nachricht gleichsam ‚mit vier Ohren gehört', also unter vier Gesichtspunkten verstanden wird:

- *Sachinhalt*: Das, worum es ‚eigentlich' geht: eine Terminabsprache, ein inhaltliches Feedback zu einem Arbeitsergebnis, eine Fehlermeldung auf einem Bildschirm usw. Der Sachinhalt wird nach dem Wahrheitsgehalt, der Relevanz und der Hinlänglichkeit beurteilt.
- *Appell*: Das, was mit der Nachricht beim Empfänger erreicht werden soll: die schnellere Fertigstellung eines Programmteils, die Bereitstellung von für die Entwicklung nötigen Informationen, das Erlernen der Nutzung einer Software u.v.a.
- *Ausdruck oder Selbstkundgabe*: Das, was in einer Nachricht über den Sender ausgedrückt wird: als was für ein Mensch stellt sich jemand in der Nachricht dar – etwa höflich und hilfsbereit (zum Beispiel unter Teamkolleg*innen), dominant und abweisend (zum Beispiel zwischen Vorgesetzt*innen und Mitarbeiter*innen) oder arrogant und

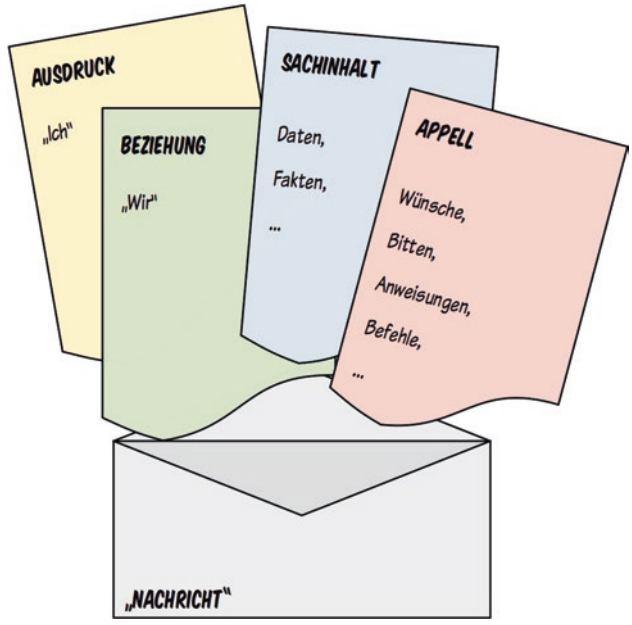

Abb. 2.3 Die vier Seiten einer Nachricht nach Schulz von Thun (eigene Darstellung)

besserwisserisch (zum Beispiel in unverständlichen Meldungen einer Software, die Kontexte voraussetzt, die Nutzer*innen nicht haben können).
- *Beziehung*: Das, was mit der Nachricht über die Beziehung von Sender und Empfänger ausgedrückt wird, das heißt, wie der Sender zum Empfänger steht: etwa ein partnerschaftliches Verhältnis zwischen Entwickler*innen und Kund*innen, ein unterwürfiges Verhältnis gegenüber Kund*innen („Der Kunde ist König"), ein dominantes Verhältnis zu Untergebenen („Wenn der Kuchen spricht, schweigt der Krümel"). Auch, ob sich Nutzer*innen von einer Software ernstgenommen oder bevormundet fühlen, kann als Problem auf Beziehungsseite benannt werden.

Zu Schwierigkeiten bei der Kommunikation kommt es, wenn Sender und Empfänger unterschiedliche Interpretationen der vier Seiten haben oder wenn die Seiten untereinander inkongruent sind:

- Beispielsweise könnten Sie auf der Sachebene Ihren Kund*innen gegenüber versichern, dass Sie an echter Kooperation und Transparenz interessiert sein, und dass Sie Fragen und Anregungen ernstnehmen und berücksichtigen. Aber wenn auf der Ausdrucks- und Beziehungsseite Ihre Wortwahl, Ihre Betonung, Ihre Körpersprache und letztlich auch Ihr tatsächliches Handeln der Sachebene widersprechen, wird das zumindest Irritation bei Ihren Kund*innen auslösen: „Sind die nur höflich, damit die unseren Auftrag kriegen?" oder „Die nehmen uns doch gar nicht richtig ernst."
- Ein anderes Beispiel wäre die Entscheidung, „von nun an agil entwickeln" zu wollen, weil man das eben heute so mache und weil dann alles viel besser sein soll. Die Umstellung auf agile Entwicklungsmodelle verlangt meist, dass sich die Kommunikations- und Kooperationskultur grundlegend ändert, und das betrifft innerste Einstellungen der beteiligten Menschen. Wenn Sie als eher traditionell sozialisierte*r Vorgesetzte*r an Hierarchien gewöhnt sind, aber eines Tages verkünden, dass von nun an Ihr Büro „immer offen" stünde und man Sie „immer ansprechen" könne, und Sie dann Mitarbeiter*innen, die das Angebot tatsächlich nutzen wollen, genervt abweisen, weil Ihre überkommene Arbeitsweise in Wahrheit gar keine spontane Kommunikation zulässt, dann stehen Sachebene, Appellseite, Ausdrucksseite und Beziehungsebene in deutlichem Widerspruch.
- Ein drittes Beispiel finden wir in Software, die oberflächlich den Eindruck macht, einfach benutzbar zu sein (übersichtlich, freundliche Gestaltung, in der Sprache der Nutzer*innen verfasste Formulierungen usw.), sich dann aber als nicht angemessen für die versprochenen oder die von Nutzer*innen zu erledigenden Aufgaben herausstellt. Der Widerspruch liegt hier auf Ausdrucksseite (Präsentation der Software) und Sachebene (Funktionalität der Software).

Schulz von Thuns Modell eignet sich vor allem, um in Kommunikationstrainings wesentliche Faktoren von Kommunikationsstörungen kennenzulernen und um rückblickend bei beobachteter Kommunikation Schwachstellen zu identifizieren. Die Anwendung auf

Softwarenutzung (wie im dritten Beispiel) ist möglich, weil Software als kommunikatives Artefakt anderer Menschen wahrgenommen wird. Als Entwickler*in sind Sie zwar abwesend, aber mitgedacht; Sie werden beispielsweise dann adressiert, wenn über „diese Programmierer!“ geschimpft oder Misstrauen über Ihre Firma ausgedrückt wird (vgl. Abschn. 6.2.2).

„Man kann nicht nicht kommunizieren“ (Paul Watzlawick)

Noch bekannter als das 4-Ohren-Modell Schulz von Thuns ist ein sogenanntes Axiom der Kommunikation, das vom amerikanischen Psychologen Paul Watzlawick (1921–2007) in den 1970ern formuliert wurde: „Man kann nicht nicht kommunizieren“ (vgl. Watzlawick et al. 2016).

Das Axiom drückt die Erkenntnis aus, dass ein Sender nicht bestimmen kann, was von einem Empfänger als kommunikativ verstanden wird und was nicht. Auch unbeabsichtigtem Verhalten können wir unterstellen, dass damit etwas ausgedrückt werden soll: Die Wahl Ihrer Kleidung („Will der mit diesem Batman-T-Shirt jugendlich wirken?“), das Tragen einer Sonnenbrille („Schützt die sich vor der Sonne, oder will sie sich abschotten?“), ein selbst als neutral empfundener, aber von anderen als missmutig gedeuteter Gesichtsausdruck („Wieso guckt der so genervt?)“ – all das sind Ausdrucksformen, mit denen die ‚Sender‘ oft gar keine Absicht verbinden, die aber trotzdem oft unterstellt wird.

Um Missverständnisse zu vermeiden, sollten Sie also gerade auch die unbewussten Aspekte Ihres Verhaltens und Ihrer Arbeit reflektieren. Hinsichtlich der kommunikativen Wirksamkeit von Software ist es hilfreich, Nutzer*innen bei der Nutzung zu beobachten und sie dabei ihre Gedanken laut aussprechen zu lassen (‚lautes Denken‘). So erkennen Sie, welche Gestaltungs- und Funktionselemente wahrgenommen und wie sie verstanden wurden (den Ablauf solcher Beobachtungen schauen wir uns in Kap. 4 bis 6 ausführlich an).

2.2.3 Kommunikation als Differenzmanagement

In Donick (2019) habe ich Kommunikationssituationen mit einem Möbiusband verglichen. Ein Möbiusband ist ein in sich verdrehtes Band, das nur eine durchgehende Seite hat, auf dem man aber trotzdem gegenüberstehen kann (Abb. 2.4). Mit dieser Metapher wollte ich ausdrücken, dass wir oft Ähnliches wollen – das ist das Kooperationsprinzip, das wir mit dem Sprachphilosophen Paul Grice jeder Kommunikation unterstellen –, aber uns trotzdem schwertun, uns zu verstehen. Das ist die Wahrnehmung: Wir stehen auf derselben Seite, und sehen uns doch weiter voneinander entfernt. Zudem sind wir umgeben von Hintergründen, die als Kontext für Verständigung dienen können, aber je nachdem, wo wir uns auf dem Möbiusband befinden, nehmen wir andere Hintergründe wahr oder blicken aus unterschiedlicher Perspektive auf sie. Ziel unserer Kommunikationsversuche ist es, sich uns auf dem Möbiusband anzunähern und eine ähnliche Perspektive zu entwickeln.

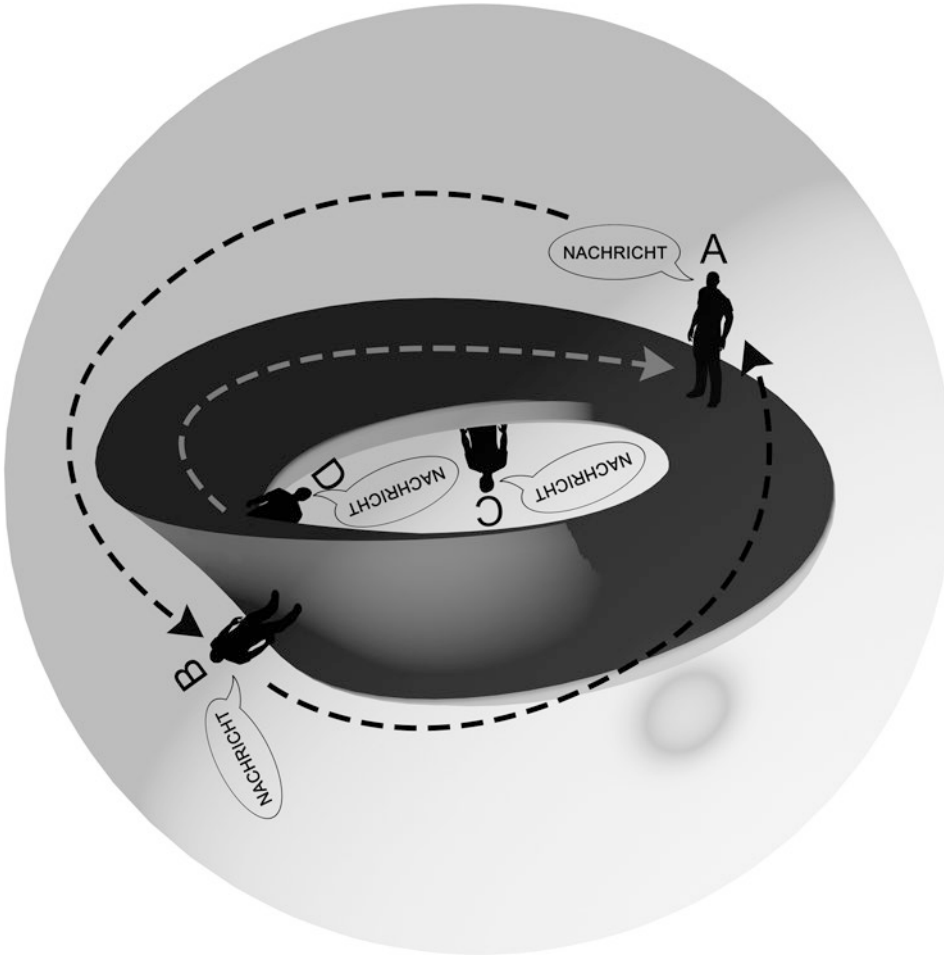

Abb. 2.4 Möbiusband als Kommunikationsmetapher (eigene Darstellung)

Die Möbiusmetapher nutze ich, um die Bedeutung der Perspektive hervorzuheben, die wir auf einander und auf Verständigungshintergründe haben. Eine formalere Sicht auf dieses Kommunikationsverständnis bietet uns die soziologische Systemtheorie Niklas Luhmanns, insbesondere in der Rezeptionsweise Dirk Baeckers, der „Form und Formen der Kommunikation“ (Baecker 2007) untersucht hat. Eine systemtheoretische Analyse, die alles Beobachtbare als System-Umwelt-Beziehungen beschreibt, ist nicht an kausalen Erklärungen interessiert, sondern an sogenannten Funktionsäquivalenten (vgl. dazu auch Abschn. 5.2). Sie sucht Alternativen zum aktuell Gegebenen. Sie fragt also nicht: Warum ist etwas so? Sondern: Wie könnte es stattdessen sein? Auf unser Anwendungsgebiet bezogen, will sie also nicht wissen, warum ein*e beobachtete Nutzer*in eine Benutzeroberfläche gut oder schlecht versteht, sondern welche alternativen Benutzungswege sich bei

der Beobachtung noch andeuten bzw. wie die Unterscheidungsprozesse aussehen, die Nutzer*innen vollziehen (vgl. Donick 2016).

Die Systemtheorie ist ein vielschichtiges Gebilde, das traditionellen wissenschaftstheoretischen Ansprüchen (etwa an Widerspruchsfreiheit und Vermeidung von Selbstreferenzialität) bewusst nicht genügen will. Baeckers Zugang vermeidet ein zu enges Verstricken in den systemtheoretischen Feinheiten, sondern verknüpft bestimmte Formen mit bestimmten Funktionen, um Ansätze für die Sinngebung von Kommunikation vorzuschlagen (Baecker spricht daher, wie schon Jakobson, von Sinnfunktionen). Dies soll im Folgenden kurz dargestellt werden, da wir uns in Kap. 3 eng an Baeckers Vorschlag orientieren werden.

Eine *form* im hier benutzten Sinne ist eine formale Darstellung, die mit den *Laws of Form* (LoF) arbeitet (vgl. Kasten).

Die Laws of Form von George Spencer-Brown

Die Laws of Form (LoF) wurden vom Mathematiker George Spencer-Brown (1923–2016) im Kontext elektrotechnischer Arbeiten entwickelt (vgl. Spencer-Brown 2011). Sie wurden bekannt, als der Physiker und Kybernetiker Heinz von Foerster 1969 eine Rezension zu den LoF verfasste.

Die LoF drücken alle algebraischen und logischen Gesetze mit Hilfe des sogenannten *cross* und des Gleichheitszeichens aus. Das *cross* hat die Form eines rechten Winkels, der eine Innenseite (links) von einer Außenseite (rechts) trennt. Zur Arbeit mit dem *cross* nennt Spencer-Brown zwei grundlegende Gesetze (Abb. 2.5).

(1) Das Gesetz der Nennung („law of calling“, ebd., S. 8) umfasst erstens die *Konfirmierung* (Wiederholung) eines eingeführten Elements und zweitens die *Kondensierung* (das Zusammenziehen) zweier Elemente auf die durch sie ausgedrückte Unterscheidung.

(2) Das Gesetz der Kreuzung („law of crossing“, ebd., S. 9) umfasst erstens die *Aufhebung* einer vorher gemachten Unterscheidung zu einer Leerstelle (void, unmarked

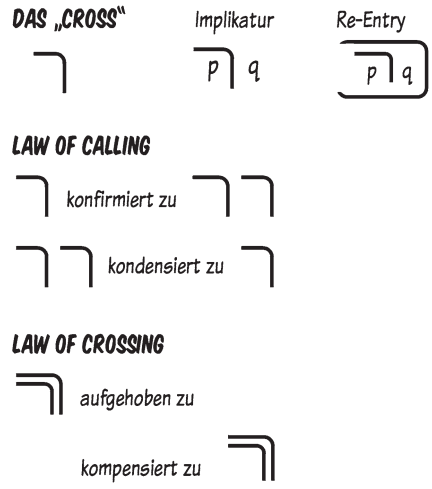

Abb. 2.5 Spencer-Browns grundlegenden Gesetze der Form (eigene Darstellung)

space) und zweitens die *Kompensation* der Leerstelle, um neue Unterscheidungen zu erlauben.

Spencer-Brown zeigt, wie mit diesen Formen die Gesetze der Algebra beschrieben, gerechnet und aussagenlogische Ausdrücke formuliert werden können (ebd., S. 91). Insbesondere aber sieht Spencer-Brown eine Möglichkeit für Selbstreferenz vor (ebd., S. 53), bei der das Ergebnis einer Operation wieder als Parameter für die Wiederholung dieser Operation verwendet wird. So kann beispielsweise ein endlos zwischen beiden Seiten oszillierendes „a impliziert b impliziert a impliziert b …“ auf elegante Weise geschrieben werden. Man kann auch sagen: „a wird vor dem Hintergrund von b unterschieden, und b vor dem Hintergrund von a“, oder „a ergibt vor dem Hintergrund von b Sinn, und b vor dem Hintergrund von a“. Das Ergebnis der Unterscheidung tritt also in die Innenseite der Form für die nächste Unterscheidung ein.

Weil durch diesen sogenannten Wiedereintritt die Selbstreferenzialität autopoietischer Systeme ausgedrückt werden kann, nehmen die LoF in Heinz von Foersters Arbeiten zur Kybernetik 2. Ordnung (von Foerster 1993), in Luhmanns Systemtheorie und in den darauf aufbauenden Arbeiten Dirk Baeckers einen großen Stellenwert ein. Sie dienen dort aber eher als mitunter etwas schematische Metapher, als dass sie mathematisch benutzt würden (zum Gewinn dieses Vorgehens vgl. Baecker 2007, S. 67 f.). In Mathematik und Philosophie selbst ist der Erkenntniswert von Spencer-Browns Arbeit umstritten. So hatte vor ihm schon der Zeichentheoretiker Charles Sanders Peirce (1839–1914) ein „sign of illation“ (Peirce 1976, S. 107) eingeführt, das Spencer-Browns *cross* sowohl optisch ähnelt, als auch dieselben logischen Relationen ausdrücken kann (Peirce hatte seinen Ansatz aufgegeben, als er seine Existential Graphs einführte, wodurch das sign of illation vergessen wurde bis ‚Peirce‘ Aufsatz 1976 neu veröffentlicht wurde; vgl. Kauffman 2001, S. 88 f.).

Baecker (2007) stellt insgesamt 22 Formen vor, die auf unterschiedliche Weise mit Kommunikation zu tun haben. Dabei nutzt Baecker stets die von links nach rechts lesbare Grundform des *a impliziert b* und markiert sie durch den Wiedereintritt als selbstreferenziell. (Mitunter gibt es auch noch *c* und *d*, aber das von Baecker genutzte Schema bleibt

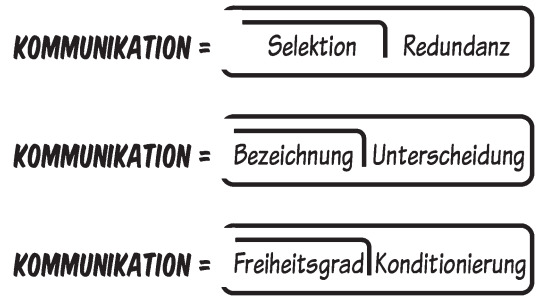

Abb. 2.6 Formen der Kommunikation nach Dirk Baecker (eigene Darstellung)

dasselbe). Kommunikation als solche drückt Baecker mit Hilfe dreier Formen aus (Abb. 2.6).

Diese Formen unterscheiden sich wie folgt:

- *Kommunikation als Differenz von Selektion und Redundanz:* Hiermit bezieht sich Baecker zunächst auf die Informationstheorie in der Tradition des Kommunikationsmodells von Shannon (1948). Aber statt einer mathematisch-statistischen Interpretation kommt es Baecker darauf an, dass ein Informationsgehalt einer konkreten Nachricht nur vor dem Hintergrund anderer möglicher Nachrichten beobachtet werden kann (Baecker 2007, S. 21); Nachrichten, aus denen ebenso gut hätte ausgewählt hätte werden. Redundanz einer Nachricht bedeute, auf diese anderen Möglichkeiten zu verweisen (ebd.). Anstatt von einer objektiv erkennbaren Wirklichkeit auszugehen, die durch Kommunikation nur abgebildet würde, betont Baecker: „Selektionen ordnen nicht die Ereignisse, Gegenstände und Zustände der Welt, sondern sie bringen diese Welt allererst hervor“ (ebd., S. 24). Ähnliches hatten wir im letzten Abschnitt schon in Bezug auf die Poststrukturalisten gehört (vgl. Kasten in Abschn. 2.2.2).
- *Kommunikation als Differenz von Bezeichnung und Unterscheidung:* Der beschriebene Selektionsvorgang (das Auswählen einer Nachricht aus einem Möglichkeitsraum auch denkbarer anderer Nachrichten) drückt Baecker als Wechselspiel aus Bezeichnung und Unterscheidung aus (ebd., S. 60). Eine Nachricht oder allgemeiner eine Bezeichnung setzt voraus, dass eine Unterscheidung getroffen wurde, und im Kontext davon unterschiedlicher Unterscheidungen sind andere Bezeichnungen nötig (ebd., S. 62). Die jeweils aktuelle Bezeichnung ist nach Baecker „die Information, mit der die Kommunikation dann arbeitet“ (ebd.).
- *Kommunikation als Differenz von Freiheitsgrad und Konditionierung:* Diese abschließende Differenz schlägt Baecker vor, weil Bezeichnung und Unterscheidung aufeinander angewiesen sind, aber das Treffen der einzelnen Unterscheidung beziehungsweise das Setzen einer Bezeichnung frei stattfinden können (ebd., S. 63). Baecker spricht davon, dass „Freiheiten nur im Kontext von Bindungen riskiert werden können, andererseits jede Bindung aber auch nur als Bindung von Spielräumen zustande kommt“ (ebd.).

Baecker weist darauf hin, dass die Seite rechts jeder Form selbst eine weitere Außenseite der Form ist; sie ist ein unmarked state (ebd., S. 62), der nicht sichtbar ist, aber mitgedacht; der offen bleibt, aber gerade dadurch die Form kontextualisiert. Die Unterscheidungen, die in einer Form ausgedrückt werden, sind somit weniger abgeschlossen, als sie rein optisch (wegen des rechts geschlossenes Rechteck) erscheinen – eine Unschärfe bleibt.

2.3 Wozu brauchen wir das?

In diesem Kapitel haben wir in einem Überblick wichtige sprachphilosophische und kommunikationswissenschaftliche Positionen kennengelernt, die alle ein Thema umkreisten: Welche Faktoren sind für erfolgreiche Verständigung von Menschen untereinander wich-

tig? Wie kann die Vagheit der Kommunikation und ihre Unwahrscheinlichkeit beschrieben werden? Dabei haben wir uns stellenweise etwas entfernt von dem konkreten Anwendungsbereich der Softwareentwicklung. Aber wie ich schon mehrfach deutlich gemacht habe: Wir sind Menschen, die für andere Menschen entwickeln; zunehmend in Teams und nach Entwicklungsmodellen, die hochkommunikativ sind. Auch unsere Produkte sind kommunikative Artefakte. Die Wörter eines Menüs, die Icons einer Symbolleiste, Animationen und Klänge, das alles muss wahrgenommen und verstanden werden.

Tab. 2.2 Bezugspunkte von Softwareentwicklung, Softwarenutzung und Kommunikation

	Bedeutung für	
Ansatz	**Entwicklungssituationen** Entwickler*innen und Kund*innen verstehen sich oft nicht auf Anhieb, und unterschiedliche Entwickler*innen untereinander auch nicht immer unbedingt.	**Nutzungssituationen** Nutzer*innen erkennen oft nicht Funktionsweise und Nutzen implementierter Programmfunktionen, oder die Nutzung geht aufgrund unklarer Gründe nicht weiter.
Modell der Sprach-funktionen nach Jakobson	**Kontext/referential:** zunächst unhinterfragte Annahmen, die man z. B. in Meetings voraussetzt und erst mal klären muss **Sender/emotive:** Eindrücke, die man beim Empfänger erweckt, z. B. Vertrauen bei Kund*innen und Motivation bei Entwickler*innen **Empfänger/conative:** was man durch die Nachricht erreichen will, z. B., dass ein*e Kund*in brauchbare User Storys schreibt. **Kontakt/phatic:** dass überhaupt Kommunikationssituationen geschaffen werden und deren Wichtigkeit für alle klar ist **Code/metalingual:** dass z. B. ein Glossar der benutzten Begriffe verwendet wird, um Missverständnisse zu reduzieren **Nachricht/poetic:** eine Form der Nachricht, die Situation und Empfänger angemessen ist.	**Kontext/referential:** „was *will* das Programm von mir?“ **Sender/emotive:** der Eindruck, den die Software erzeugt, wie *Joy of Use* oder Überforderung; oder der Eindruck, den nutzungsbegleitende Äußerungen der Nutzer*innen erzeugen, wie Frustration oder Aggression **Empfänger/conative:** dass z. B. ein Element der Benutzeroberfläche die ‚richtige‘ Aktion bei den Nutzer*innen motiviert **Kontakt/phatic:** dass Nutzer*innen Äußerungen der Software wahrnehmen, und dass die Software auf Eingaben reagiert **Code/metalingual:** dass Software die Sprache der tatsächlichen Nutzer*innen verwendet, statt IT-Fachsprache oder Fachsprache des Managements der Nutzer*innen **Nachricht/poetic:** dass Software eine Form hat, die ganz allgemein angenehm ist

(Fortsetzung)

Tab. 2.2 (Fortsetzung)

	Bedeutung für	
4-Ohren-Modell nach Schulz von Thun	**Sachinhalt:** z. B. stehen benötigte Daten, Fakten, Absprachen, Modelle, usw. nicht allen Beteiligten vollständig zur Verfügung **Ausdruck:** z. B. präsentieren sich Entwickler*innen gegenüber Kund*innen als kompetent und vertrauenswürdig oder inkompetent und überfordert **Appell:** z. B. wünschen sich Kund*innen von Entwickler*innen, schnell, kostengünstig und ohne Aufwand für die Kund*innen zu arbeiten **Beziehung:** z. B. eine respektvolle Arbeitsbeziehung der beiden Programmierer*innen beim Pair Programming; eine unbewusste Herr-Diener-Mentalität, die manche Kund*innen einem entgegenbringen	**Sachinhalt:** z. B. sind Fehlermeldungen nicht verständlich, weil sie sich auf Gegenstände beziehen, die den Nutzer*innen nicht bekannt sind **Ausdruck:** z. B. präsentiert sich die Software als transparent, lernförderlich, unterstützend, ohne bevormundend zu sein **Appell:** z. B. will ein*e Nutzer*in durch hektische Mausklicks erreichen, dass eine scheinbar ‚eingefrorene' Software wieder reagiert **Beziehung:** z. B. Meldungen der Software, die eine gewisse Arroganz der Entwickler*innen auszudrücken scheinen
Kommunikation als 2-Seiten-Formen nach Dirk Baecker	**Selektion/Redundanz:** z. B. die Erstellung von User Storys als Auswahl aus einem Raum möglicher Anforderungen; die Priorisierung der User Storys **Bezeichnung/Unterscheidung:** z. B. die Erstellung eines begrifflichen Glossars, das unterschiedliche Verstehenshintergründe annähern soll **Freiheitsgrad/Konditionierung:** z. B. Agile Entwicklungsmodelle vs. starre Alternativen	**Selektion/Redundanz:** z. B. die Auswahl einer als brauchbar erscheinenden Programmfunktion aus der Vielfalt aller Optionen **Bezeichnung/Unterscheidung:** z. B. die Schwierigkeit, die Bedeutung einer Fehlermeldung im Kontext der Software und der Situation zu verstehen **Freiheitsgrad/Konditionierung:** z. B. die Entscheidung für einen mehrerer möglicher Bedienwege

Wir können aus den vorgestellten Modellen und Theorien einiges für unsere praktische Arbeit mitnehmen. Die wesentlichen Gedanken dazu habe ich in der Übersicht in Tab. 2.2 dargestellt.

Auf den letzten Aspekt, nämlich die Bedeutung der 2-Seiten-Formen von Kommunikation für die Beziehung von Nutzer*innen und Software, kommen wir nun zu sprechen (Kap. 3).

Literatur

Baecker, Dirk: Form und Formen der Kommunikation. Frankfurt/Main 2007.

Bühler, Karl: Sprachtheorie: Die Darstellungsfunktion der Sprache. Stuttgart 1965.
Clark, Herbert H.: Using language. Cambridge 1996.
Donick, Mario: „Offensichtlich weigert sich Facebook, mir darauf eine Antwort zu geben“: Strukturelle Analysen und sinnfunktionale Interpretationen zu Unsicherheit und Ordnung der Computernutzung. Hamburg 2016.
Donick, Mario: Die Unschuld der Maschinen. Technikvertrauen in einer smarten Welt. Wiesbaden 2019.
von Foerster, Heinz: Wissen und Gewissen. Frankfurt/Main 1993.
Herrmann, Thomas: Kommunikation und Kooperation. In: Schwabe G./Streitz N./Unland R. (Hrsg.): CSCW-Kompendium. Berlin/Heidelberg 2001, S. 15–25.
Jakobson, Roman: Linguistics and Poetics. In: Sebeok, Thomas A.: Style in Language. MIT Press, Cambridge, MA, 1960, 350–377 (Seitenangaben in Zitaten beziehen sich auf die digitale Version: https://pure.mpg.de/pubman/faces/ViewItemFullPage.jsp?itemId=item_2350615_3).
Kauffman, Louis H.: The Mathematics of Charles Sanders Peirce. In: Cybernetics & Human Knowing, Vol. 8, No. 1–2, 2001, S. 79-110.
Linke, Angelika / Nussbaumer, Markus / Portmann, Paul R.: Studienbuch Linguistik. Tübingen 2004.
Peirce, Charles S.: The Logical Algebra of Boole. In: Eisele, Carolyn (Hrsg.): The New Elements of Mathematics. By Charles S. Peirce. Den Haag/Paris, 1976, S. 106–115.
Rau, Karl-Heinz: Agile objektorientierte Software-Entwicklung. Schritt für Schritt vom Geschäftsprozess zum Java-Programm. Wiesbaden 2016.
Schulz v. Thun, Friedemann: Miteinander Reden. Band 1: Störungen und Klärungen. Allgemeine Psychologie der Kommunikation. Reinbek bei Hamburg 2010.
Shannon, Claude E: A mathematical theory of communication. Bell Systems Technical Journal 27, 1948, S. 379–423 & S. 623–656.
van Someron, Maarten W./Barnard, Yvonne F./Sandberg, Jacobijn A.C.: The Think Aloud Method. A practical guide to modelling cognitive processes. London 1994.
Spencer-Brown, George: Laws of Form. Leipzig 2011.
Sperber, Dan / Wilson, Deidre: Relevance. Communication and Cognition. Oxford 1995.
Watzlawick, Paul/Beavin JH/Jackson D: Menschliche Kommunikation: Formen, Störungen, Paradoxien. Bonn 2016.
Wikipedia-Eintrag „Extreme Programming“ URL: https://de.wikipedia.org/wiki/Extreme_Programming (letzter Abruf: 10.09.2019).

Über die Beziehung von Nutzer*in, Software und Nutzungssituation

3

Inhaltsverzeichnis

3.1 *Ziel von Entwicklung:* Relevante *Werkzeuge herstellen*

3.1.1 Relevanz als Kommunikationsmerkmal

Wenn Sie Software entwickeln, die von Nutzer*innen verstanden und erfolgreich eingesetzt werden soll, dann muss diese Software den Anforderungen der Nutzer*innen in der jeweiligen Nutzungssituation genügen. Sie muss für die Nutzer*innen *relevante Leistungen* erbringen und die Ausgaben der Software müssen diese Relevanz ausdrücken. Um das sicherzustellen, ist es hilfreich, wenn Sie sich über grundlegende Merkmale der Beziehung von Nutzer*innen und Software im Klaren sind. Denn wie in Kap. 1 schon angesprochen wurde, entsprechen echte Nutzer*innen keinen Idealtypen, wie man sie sich in Modellen ausmalt. Genauso sind echte Nutzungssituationen vielfältiger und unvorhersehbarer, als Sie sie in einem Aufgabenmodell oder Kontextmodell abbilden könnten (vgl. Suchman 2007; Donick 2016). Wie wir am Ende des letzten Kapitels gesehen haben, ist

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8_3

Kommunikation der Versuch, aus einer Reihe von Möglichkeiten diejenigen auszuwählen, die sich in der Situation hoffentlich bewähren und damit andere Möglichkeiten auszuschließen (vgl. Abschn. 2.2). Daher ist jede Produktentwicklung – selbst unter angenommenen unrealistisch idealen ökonomischen Bedingungen – immer nur eine Annäherung an die Nutzer*innen und Situationen. Wir könnten sagen:

> Von der Kommunikation her gedacht, ist der Kern jeder Gestaltungsaufgabe, Relevanzprozesse sicherzustellen.

Was heißt das?
In der Linguistik ist „Sei relevant!" eine von vier Konversationsmaximen (vgl. Kasten), die vom Sprachphilosophen Herbert Paul Grice (1913–1988) genannt wurden. Sie ist ein Teil des übergeordneten Kooperationsprinzips, das jeder Kommunikation zugrunde liegt: „Kommunikation kann nicht zustande kommen, wenn die Beteiligten nicht wenigstens ein minimales gemeinsames Interesse haben" (Linke et al. 2004, S. 220). Ob aber etwas relevant ist, ist in erster Linie eine Beobachtung des Empfängers – keine Entscheidung des Senders. Ein Sender kann sich um Relevanz für einen Empfänger bemühen, aber ob dies auch so ankommt, ist eine andere Frage. Es ist daher treffender, wie Luhmann von geschlossenen Systemen auszugehen, die nur ihre Umwelt beobachten können und nur für sich selbst mit Irritationen aus der Umwelt umgehen müssen.

Kooperationsprinzip und Konversationsmaximen (Herbert Paul Grice)

Wenn wir miteinander kommunizieren, unterstellen wir uns gegenseitig, dass wir kooperieren wollen – nicht dahingehend, dass wir hinterher unbedingt dieselbe Meinung vertreten oder dasselbe Handlungsziel verfolgen wollen, aber zumindest insoweit, *dass* wir kommunizieren wollen. Für die sprachliche Kommunikation bezeichnet Grice dies als Kooperationsprinzip (vgl. Grice 1991; Linke et al. 2004, S. 223). Für die Erfüllung des Kooperationsprinzips gibt es vier sogenannte Maximen:
Maxime der Quantität: Sag so viel wie nötig, aber so wenig wie möglich. (vgl. ebd.).
Maxime der Qualität: Sag nur Dinge, die du für wahr hältst bzw. mache deutlich, wie wahrscheinlich das Gesagte ist. (vgl. ebd.)
Maxime der Relation: „Sei relevant." (ebd.)
Maxime der Modalität: „Sag deine Sache in angemessener Art und Weise und so klar wie nötig". (ebd.)

Für zu entwickelnde Software sind die Konversationsmaximen interessant, weil auch Nutzer*innen einer Software als kommunikatives Artefakt (das von Entwickler*innen hergestellt wurde) zunächst unterstellen, quantitativ und modal angemessen zu sein, qualitativ korrekte Ergebnisse zu liefern und für die Nutzungssituation relevant zu sein – sonst hätten sie sich nicht für die Software entschieden (oder hätten diese nicht von Vorgesetzten als Werkzeug erhalten).

Grice' Konversationsmaximen beziehen sich insbesondere auf stark situationsabhängige Kommunikation, bei der das bloße kommunikative Material (Sprache, aber wir

können auch andere Medien dazunehmen) sowie leicht erschließbare Wissensbestände nicht ausreichen, auf den Sinn des Ausgedrückten zu schließen. Das sind die sogenannten *konversationellen Implikaturen*. Im Alltag fallen uns solche Schlussprozesse kaum auf. Um sie zu demonstrieren, diskutieren Linke et al. (2004) die Situation eines fortgeschrittenen Abends, bei dem Gäste eines Gastgebers gerade ein neues Diskussionsthema begonnen haben, worauf der Gastgeber plötzlich fragt: „Sagt mal, habt ihr nicht davon gesprochen, dass ihr morgen früh raus müsst?“ (ebd., S. 221). Insbesondere die Maxime der Relation ist in der Situation verletzt (die Frage passt nicht zum gerade begonnenen Diskussionsthema) – und trotzdem verstehen wir, was gemeint ist: Dass der Gastgeber nicht mehr auf das Diskussionsangebot eingehen möchte, weil es spät ist, und dass es für die Gäste langsam an der Zeit ist, zu gehen (ebd.).

Problematisch werden konversationelle Implikaturen, wenn sie in Situationen verwendet werden, die keine sinnvollen Schlussprozesse zulassen – sei es, weil dafür keine Zeit ist, oder sei es, weil die nötigen Interpretationsverfahren von den beteiligten Menschen nicht erbracht werden können, etwa wegen kultureller Unterschiede (was nicht nur Nationalkulturen meint, sondern auch Unterschiede in den Arbeitskulturen verschiedener Berufsfelder und Disziplinen).

Für ein System, das seine Umwelt beobachtet, ist ein Reiz relevant, wenn der Reiz wahrgenommen wird und einen sogenannten Kontexteffekt auslöst (vgl. Sperber und Wilson 1995, S. 39). Kontext ist die systemeigene Repräsentation aller seiner Beobachtungen und Annahmen. Ein Kontexteffekt tritt ein, wenn der von außen kommende Reiz an die vorhandenen Repräsentationen anknüpfen kann. War der Aufwand zur Verarbeitung des Reizes gering, aber der Kontexteffekt groß, dann war der Reiz relevanter, als wenn der Aufwand hoch und der Effekt gering war (ebd.). Vielleicht kennen Sie die Grundidee aus der Cognitive Load Theory von John Sweller (1988). Ich beziehe mich in diesem Abschnitt vorwiegend auf die Relevanztheorie der Linguisten Dan Sperber und Deidre Wilson. In Bezug auf Software spielt Relevanz eine Rolle einerseits hinsichtlich der Leistung der Software und andererseits hinsichtlich der kommunikativen Ausdrücke der Software.

3.1.2 Software und Relevanz

Relevante Leistung von Software

Als Entwickler*in stellen Sie Software für andere Menschen bereit. Damit diese Menschen Ihre Software verwenden, geben Sie in irgendeiner Weise ein Leistungsversprechen ab – neben formalen Vereinbarungen mit Ihren direkten Kund*innen (Lasten- bzw. Pflichtenheft, User Storys) kann es sich auch um eine Liste von Funktionsmerkmalen (Features) auf einer Website sein, die sich an anonyme Kund*innen richtet. Sie versprechen jemandem, dass Ihre Software eine Leistung erbringt. Sie müssen nicht nur sicherstellen, dass dieses Versprechen aus Ihrer Sicht oder für eine*n neutrale*n Beobachter*in rein funktional erfüllt ist, sondern auch, dass Ihre Kund*innen oder Nutzer*innen erkennen, dass die Leistung

erbracht wird. Sie müssen also deutlich machen, was Ihre Software tut (das betrifft Geschäftsprozesse) und dass es das ist, was gewünscht ist (das betrifft Ergebnisse der Geschäftsprozesse und deren Darstellung auf der Benutzeroberfläche).

In diesem Zusammenhang spricht man oft von Akzeptanztests (vgl. http://www.agile-modeling.com/artifacts/acceptanceTests.htm). Darin werden mehrere Testfälle definiert (bei agilen Methoden basierend auf den User Storys) und – in der Regel automatisiert oder halb-automatisch (ebd.) – durchgeführt. Zu einer Definition gehören Anfangsbedingungen, die auszuführenden Aufgaben und die erwarteten Ergebnisse. Die Definition wird durch die Kund*innen der Entwickler*innen vorgenommen, gegebenenfalls mit Hilfe ersterer; Testframeworks wie FitNesse (fitnesse.org) unterstützen dabei.

Beispielsweise schreiben Sie eine Software, mit der ein Verein seine Mitglieder*innen verwalten kann. Eine User Story besagt, dass automatisiert eine Zahlungserinnerung an registrierte Mitglieder*innen gemailt werden soll, wenn diese länger als zwei Wochen mit dem Mitgliedsbeitrag im Rückstand sind. Für den Akzeptanztest dieses Geschäftsprozesses wird zuerst ein Datenbank-Zustand geschaffen, der ein Testen dieser User Story erlaubt (das heißt, es werden Nutzer mit unterschiedlichen Zahlungszielen angelegt und ein Ist-Datum festgelegt). Als erwartetes Ergebnis wird definiert, welche der in der Datenbank erfassten Personen zu dem festgelegten Ist-Datum eine Zahlungserinnerung erhalten sollten und welche nicht. Dann wird getestet, ob die richtigen Personen die Mail erhalten (und die nicht betroffenen keine bekommen). Wenn das Ergebnis des Tests mit dem erwarteten Ergebnis übereinstimmt (das beurteilen Ihre Kund*innen, für die Sie die Software entwickeln), ist der Akzeptanztest für diese User Story bestanden. Die Software ist – zumindest in diesem Teilbereich – offenbar relevant für Ihre Kund*innen. Und da Sie viele User Storys haben und die auch nach Priorität sortiert sind, haben Sie am Ende aller erfolgreichen Akzeptanztests idealerweise eine stets relevante Software.

Nun sind aber die Personen, mit denen Sie User Storys erarbeiten, die Definitionen für Akzeptanztests erstellen und die Ergebnisse beurteilen, nicht immer dieselben, die im Alltag tatsächlich mit der Software arbeiten müssen, oder, wenn es Endbenutzer*innen sind, sind es oft schon ‚kundige' Nutzer*innen. Und genau da ist die Herausforderung – auch am Entwicklungsprozess unbeteiligte Personen und ‚unkundige' Nutzer*innen (vgl. dazu noch mal den Kasten in Kap. 2) müssen erkennen können, dass die Software eine relevante Leistung erbracht hat – und zwar in einer Situation, die komplexer ist, als durch in User Storys runtergebrochene Einzelteile erfasst ist. Dies kann durch automatisierte Akzeptanztests allein *nicht* sichergestellt werden.

Relevante kommunikative Ausdrücke der Software

Stellen wir uns eine Meldung vor, die Ihre Software unter bestimmten Bedingungen auswirft – eine Fehlermeldung, eine Information oder eine Frage. Zum Beispiel werden Sie informiert, wenn die automatische Mail an die säumigen Vereinsmitglieder*innen rausgegangen ist. Sie als Entwickler*in definieren die Bedingungen, unter denen die Meldung erscheint, und Sie entscheiden über Form und Inhalt der Meldung. Ihre Nutzer*innen kennen diese Bedingungen nicht immer und haben mitunter Schwierigkeiten, die Form zu

verstehen (zum Beispiel die verwendete Sprache). Darum haben sie oft Schwierigkeiten, die Relevanz der Meldung zu erkennen. Deshalb werden Meldungen gerne ungelesen weggeklickt, was unter Umständen zu Folgeproblemen führen kann. Das möchte man gern vermeiden. Damit aber die Nutzer*innen die Relevanz der Meldung erkennen, muss die Meldung so eingesetzt und formuliert sein, dass die Nutzer*innen sie an vorhandenes Vorwissen und an die jeweilige Situation anknüpfen können. Gelingt dies, dann hat die Fehlermeldung einen Kontexteffekt ausgelöst – sie war relevant.

Der Grad der Relevanz kann sich aber unterscheiden; eine Meldung, die eigentlich sofortiges Handeln verlangen würde, kann auch als bloße Information eingestuft werden. Die Art und Weise, wie die Meldung erscheint und was sie inhaltlich ausdrückt, hat darauf einen großen Einfluss. Wann und wo sollte zum Beispiel die Meldung bezüglich der automatischen Mitgliedszahlungs-E-Mails erscheinen? In einem Popup-Fenster nach dem Senden (was einen zeitlichen Zusammenhang hätte, aber auch einen möglichen anderen Arbeitsvorgang stört)? Zum nächsten Programmstart (was vielleicht dazu führt, die Meldung erst sehr spät wahrzunehmen?) In einem Nachrichtenbereich der Software (dessen Existenz vielleicht wie „Kanonen auf Spatzen schießen“ wirkt, weil es außer dieser einen Meldung sonst nicht viele Nachrichten anzuzeigen gibt)? In einer vom jeweiligen Betriebssystem vorgesehenen Benachrichtigungsposition (was verwirren könnte, weil man dort keine Ausgaben der Software erwartet, sondern nur betriebssystemrelevante Informationen)? Solche Fragen können Sie zwar bei der Definition von User Storys festlegen, und Sie können dafür auch Akzeptanztests durchführen, aber ob die Maxime der Relevanz erfüllt ist, klären Sie am besten durch die Beobachtung von Nutzer*innen.

Förderung von Relevanz als Entwicklungsfrage

Zusammengefasst sollten Sie sich als Entwickler*in eine Frage stellen (und Sie können hierüber gern kurz aus Ihrer Erfahrung heraus nachdenken, bevor Sie weiterlesen):

> Wie können Sie fördern, dass die richtigen Ausdrücke und Aktivitäten Ihrer Software in der jeweils richtigen Situation für möglichst alle von Ihnen angesprochenen Nutzer*innen als relevant erscheinen?

Wegen der Situations- und Personenabhängigkeit gibt es keine generelle Antwort auf diese Frage. Was für Situation A und Nutzer*in 1 ‚funktioniert‘, kann für Situation B und Nutzer*in 2 schiefgehen. Deswegen ist es so wichtig, dass Sie sich als Beobachter*in 2. Ordnung installieren und Nutzer*innen in möglichst realistischen Situationen beim Umgang mit der Software beobachten. Bei so einer Beobachtung erhalten Sie aus erster Hand Rückmeldungen über Relevanzprozesse, die von Ihrer Software angeregt oder nicht angeregt werden.

Nun können wir Relevanz nicht direkt beobachten – wir können ja nicht in die geschlossenen Systeme bzw. in die Köpfe von Nutzer*innen reinschauen. Wenn Sie Ihre

Nutzer*innen allerdings die Nutzung kommentieren lassen (‚lautes Denken'), finden Sie in den sprachlichen Strukturen dieser Kommentare Hinweise auf Relevanz – in den Themen (worüber man spricht), in der Kohärenz (im Zusammenhang der Kommentare) und in den Spuren struktureller Kopplungen, die zwischen System „Nutzer*in" und Umwelt „Software" entstehen, bestehen oder vergehen. Wie Sie aufgezeichnete Kommentare nach diesen drei Aspekten analysieren und interpretieren können, lernen Sie in Kap. 5 und 6. Um uns darauf vorzubereiten, worauf wir uns mit diesen Beobachtungen einlassen, schauen wir uns zuvor an, wie wir die Beziehung von Nutzer*in, Software und Situation mit sechs zentralen Begriffen umreißen können.

3.2 Zweiseiten-Formen der Softwarenutzung

Die folgenden Begriffspaare habe ich in meiner Dissertation (Donick 2016) vorgestellt und sie als sogenannte sensibilisierende Konzepte (vgl. Kap. 4) verwendet, um empirische Studien zur Computernutzung vorzubereiten. Grundlage waren einige von Baeckers Formen der Kommunikation (2007, vgl. Kap. 2). Ich hatte diese Formen wie Klassen bei objektorientierter Programmierung verwendet: Als Vorlage, aus denen ich dann für meine Zwecke konkrete Instanzen abgeleitet hatte. Genauso gehe ich auch in vorliegendem Buch vor, denn Baeckers Formen können eine große Allgemeingültigkeit beanspruchen. Was ich hier für Software und die Beziehung von Software zu Nutzer*innen und Situationen tue, ist natürlich auch für viele andere Kommunikationszusammenhänge möglich, etwa um die Kommunikation innerhalb von Teams zu untersuchen.

3.2.1 Reproduktion und Störung – Nutzer*innen als Systeme

In Fortführung von Luhmann definiert Baecker (2007) ein System als Differenz von Reproduktion und Störung (Baecker 2007, S. 153). Systeme bestehen, so lange sie sich selbst autopoietisch erhalten. Den Begriff der Autopoiesis hat Luhmann aus der Biologie übernommen (vgl. Maturana 2000; Luhmann 1996). Beispielsweise ist die Struktur eines lebenden Organismus derart, dass er Operationen der Zellerzeugung ermöglicht, die wiederum den Erhalt des Organismus erlauben (vgl. Baraldi et al. 1998, S. 29 f.). Luhmann hat diesen Gedanken auf soziale Systeme angewandt: Eine Gesellschaft ist strukturell so determiniert, dass sie Kommunikation produziert, die an andere Kommunikation anschließen kann (ebd., S. 30).

Nun findet die Reproduktion eines Systems nicht ungestört statt, sondern wird laufend durch Einflüsse aus der Umwelt irritiert. Diese Einflüsse können sich als Leistungsbeziehung zeigen, die man strukturelle Kopplung nennt. Sie können auch dazu führen, dass sich ein System weiter ausdifferenziert (man sagt manchmal vereinfachend, es passt sich der Umwelt an). Darum hat Baecker zur Beschreibung des Systems die Zweiseiten-Form aus Reproduktion und Störung gewählt – beides setzt sich voraus, ist aber auch vor weiteren (in der Form unmarkierten) Hintergründen verschieden.

In Donick (2016) habe ich die Nutzer*innen von Computern als Systeme betrachtet – als eine besondere Erscheinungsform von ‚menschlichen Systemen', die bei Luhmann eher allgemein als psychische Systeme auftauchen. Ich gehe davon aus, dass die Zuschreibung eines System-Umwelt-Verhältnisses ebenfalls ‚nur' eine Beobachtung von außen ist, und dass man daher dieselbe menschliche Entität mal als diese, mal als jene Art von System beobachten kann (häufiger spricht man wohl von Rollen): als Programmierer*in, als Elternteil, als Sportler*in, als Autofahrer*in und so weiter. In jeder dieser Rollen lässt sich die Entität anhand bestimmter struktureller Parameter, die man von außen beobachten kann, als System von einer Umwelt abgrenzen. Als Programmierer*in beobachte ich Sie zum Beispiel beim Schreiben von Quellcode, beim Sprechen über Code, beim Debuggen, vielleicht auch bei Selbstinszenierungsprozessen, die über die eigentliche Programmiersituation hinausweisen, zum Beispiel Kleidung, Slang, ‚typische' Verhaltensweisen. Kommen alle diese Merkmale in bestimmten Situationen für eine*n Beobachter*in zusammen, erzeugen sie ein so abgrenzbares Bild, dass man von einem System sprechen kann. Sind diese Merkmale plötzlich nicht mehr zu beobachten bzw. stehen bei der Entität plötzlich ganz andere Merkmale im Vordergrund, dann ist die Entität in einer anderen Rolle bzw. als andere Art von System zu beobachtbar. Dies ist der unsichtbare *unmarked space* außerhalb der Form.

Für Nutzer*innen als Systeme sehe ich in Donick (2016) vor allem die fortwährende Produktion softwarebezogener Anschlusshandlungen als entscheidend an. Aufbauend auf die Untersuchungen von Suchman (2007), die herausgearbeitet hat, dass Nutzer*innen sich unbewusst stets fragen, ‚wie es weitergeht' („What's next?"), sehe ich Menschen dann als Nutzer*in an (bzw. beobachte sie als entsprechendes System), wenn ihre Aktivitäten eine strukturelle Kopplung zu ihrer Software-Umwelt aufbauen oder aufrechterhalten (Abb. 3.1). Sind dahingehend keine Spuren mehr beobachtbar (keine Spuren, die auf strukturelle Kopplung hinweisen), dann ist die Entität nicht mehr als Nutzer*in-System beobachtbar, sondern in einer anderen Rolle – als Teil eines sozialen Interaktionssystems etwa, wenn die Person bei einem technischen Kundendienst anruft (vgl. Donick 2019), oder als Katzenfreund*in, wenn die Katze auf die Tastatur gesprungen ist und die Arbeit dadurch für eine deutlich erkennbare Zeitspanne unterbricht.

Entitäten reproduzieren sich als **Nutzer*in**-System, indem sie Teil des Verhältnisses Eingabe – Verarbeitung – Ausgabe sind. Wenn Sie auf eine Bildschirmausgabe mit einer passenden Eingabe antworten (die Ausgabe also relevant für Sie war), dann sind Sie als Nutzer*in beobachtbar. Sie haben eine Anschlusshandlung erzeugt. Durch Ihre Handlungen grenzen Sie sich zu einer Umwelt ab, in der diese Handlungen keine Bedeutung haben. Beispielsweise ist ein Mausklick nur im Kontext einer Nutzer*in-Software-Beziehung

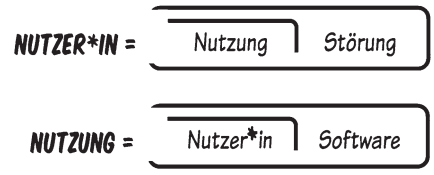

Abb. 3.1 Nutzer*in = [Reproduktion] Störung

überhaupt sinnhaft einzuordnen. Die Beobachtung, dass Sie eine Maustaste gedrückt haben, ist struktureller Hinweis auf den Aufbau oder das Bestehen einer strukturellen Kopplung zwischen Ihnen als System und einem Computer(programm) als Ihrer Umwelt. Reagiert der Computer nicht auf den Mausklick, oder in ‚falscher' Art und Weise, kann dies als Hinweis auf den Bruch einer bestehenden Kopplung interpretiert werden.

3.2.2 Form und Funktion – Software als Design

Software ist ein technisches System, aber kein autopoietisches, sich selbst erhaltendes System im Sinne der Luhmann'schen Systemtheorie. Software entsteht nicht (zumindest heute noch nicht) aus ihren eigenen Unterscheidungen einer Umwelt, sondern Software ist ein gestaltetes technisches Artefakt. Dirk Baecker definiert Design als Differenz von Funktion und Form (Baecker 2007, S. 265). Das bedeutet, dass Funktionen bestimmte Formen implizieren, und dass bestimmte Formen auch auf mögliche Funktionen zurückwirken. Beides geschieht vor dem Hintergrund weiterer (in der Form nicht markierter) Hintergründe, zum Beispiel der Nutzungssituation. Der bekannte Satz „form follows function" drückt also nur eine von drei gleichberechtigten Perspektiven auf das Verhältnis beider Begriffe aus (Funktionen implizieren Formen; Formen implizieren Funktionen; ein Form-Funktions-Verhältnis impliziert einen Hintergrund, vor dem wir das Vorhandensein so eines Verhältnisses beobachten können).

In diesem Sinne habe ich aus Baeckers „Design"-Form die Form **Software** abgeleitet, als Differenz von Funktion und Benutzeroberfläche (Donick 2016, S. 125, Abb. 3.2).

Wir nutzen Software, um von ihrer Funktionalität zu profitieren; wir tun das, indem wir mit einer Benutzeroberfläche auf die Software zugreifen. Die Oberfläche erlaubt uns einen bestimmten Blickwinkel auf die implementierte Funktionalität; sie legt manche Funktionalität offen und versteckt andere vor uns.

3.2.3 Plan und Abweichung – Nutzungsziele in der Nutzungssituation

Für Gestaltung und Nutzung von Software gleichermaßen spielen Pläne und Abweichungen von Plänen eine besondere Rolle. Auf Entwicklerseite etwa kann man folgendes sagen:

- Die Definition von Anforderungen, Geschäftsprozessen, Modellen usw. fällt alles unter das Erstellen von Plänen;
- die Implementierung setzt Pläne in Algorithmen und Datenstrukturen um;
- die Dokumentation hält Pläne schriftlich fest;

Abb. 3.2 Software = [Funktion] Benutzeroberfläche

- Tests belegen die Passung von Plänen und Implementierung;
- an Nutzer*innen gerichtete Handbücher, Hilfetexte und Tutorials erklären den Plan hinter der Software.

Die Nutzung folgt ebenfalls Plänen, auch wenn dies weniger explizit gemacht sein muss als bei der Entwicklung:

- Nutzer*innen verfolgen ein Ziel und haben gewisse Annahmen, wie sie mit der Software dieses Ziel erreichen können;
- diese Annahmen können schon vorher da sein und erst zur Entscheidung für eine Software führen;
- Annahmen können aber auch durch das Erlernen der Software und durch ihre fortwährende Nutzung gebildet oder modifiziert werden;
- zudem kann die jeweilige Situation die Annahmen verändern.

Wir haben es also auf beiden Seiten mit Plänen und deren Umsetzung zu tun. Pläne sind eine Erwartung, wie etwas sein wird oder sein soll. Abweichungen von Plänen sind Enttäuschungen der Erwartung (was nicht per se negativ sein muss; es gibt auch Enttäuschungen, die wir begrüßen können). Dirk Baecker drückt Erwartung selbstreferenziell in der gleichnamigen Zweiseiten-Form aus und definiert diese als Differenz von Erwartung und Enttäuschung (Baecker 2007, S. 89). Ich habe dies modifiziert zur **Ziel**-Form als Differenz von Plan und Abweichung (Donick 2016, S. 137, Abb. 3.3).

Dass wir oft von einmal gemachten Plänen abweichen, und dass ein zu starres Festhalten an Plänen Schwierigkeiten bereitet, zeigen nicht nur kommunikationswissenschaftliche Beobachtungen (wie sie zum Beispiel in der Ethnomethodologie angestellt werden, vgl. Suchman 2007), sondern auch die offenbar erkannte Notwendigkeit, von starren Modellen für die Softwareentwicklung wegzukommen und diese durch agilere Ansätze zu ersetzen. Hier wurde erkannt, dass Pläne flexibel an sich ändernde Situationen angepasst werden müssen, wenn man diesen Situationen weiterhin gerecht werden will. Nur gilt dies eben nicht nur für die Entwicklung, sondern auch für die Nutzung von Software.

Einen Plan zu machen, heißt, sich auf eine Reihe von Optionen aus einem umfassenderen Möglichkeitsraum festzulegen. Ein System muss dann nicht quasi momenthaft von Unterscheidung zu Unterscheidung operieren, sondern kann zukünftige Unterscheidungen vorwegnehmen. Dies entlastet das System für den aktuellen Moment und schafft Kapazitäten, andere Unterscheidungen vorzunehmen. Die Konsequenz einer enttäuschten Erwartung (einer Abweichung vom Plan) bedeutet, dass das System entscheiden muss, ob es auf die Abweichung in irgendeiner Weise reagieren muss (also den Plan anpassen), oder ob es mit der Abweichung leben kann (trotz der Abweichung mit dem Rest des Plans fortzufah-

Abb. 3.3 Ziel = [Plan] Abweichung

ren). In der Systemtheorie spricht man hier auch von der Ausdifferenzierung des Systems, oder kurz: seiner Evolution, wenn es sich nach Irritationen der Umwelt an neue Umstände anpasst.

3.2.4 Einfaches Beispiel aus der Praxis

Um besser zu verstehen, wie die genannten Formen in der Praxis zusammenwirken, zeige ich in Beispiel 3.1 einen Transkriptausschnitt aus Donick (2016, S. 122). Damit erhalten Sie schon einen Vorgeschmack darauf, was Sie mit den Verfahren, die in Kap. 4, 5 und 6 vorgestellt werden, über Softwarenutzung herausbekommen können.

Das Beispiel ist ein Ausschnitt aus der Beobachtung des Umgangs mit einer Adressdatenbank. Diese Datenbank sollte dazu dienen, schnell und einfach Rundmails zu versenden. Die Software war zum Zeitpunkt der Beobachtung noch in Entwicklung. Die beobachtete Nutzerin war eine ‚echte' avisierte Endnutzerin der Datenbank und an der Entwicklung selbst nicht beteiligt. Sie wurde gebeten, erste Tests der Grundfunktionen vorzunehmen. Sie sollte einige Daten zu Personen eintragen, einen Mailtext eingeben und diesen dann an die vorher eingetragenen Personen versenden. Um zu beobachten, was Software und Nutzerin bei Schwierigkeiten tun, war die Aufgabe so formuliert, dass es zu Problemen kommen musste: Der Nutzerin wurden keine E-Mail-Adressen mitgeteilt, sodass sie den Aufgabenteil „Versenden" nicht erfüllen konnte.

Beispiel 3.1

[…]
1 also ich geh auf Mails versenden
2 aber da . passiert nichts weiter [unverständlich] seltsam .
3 das is n bisschen lustig
4 weil ich hab ja keine E-Mail-Adresse von diesen Leuten
[…]

Bis zum Beginn des Ausschnitts kam die Nutzerin mit der Datenbank gut zurecht. In systemtheoretischen Begriffen: Das **Nutzer*in**-System war über einen längeren Zeitraum als strukturell gekoppelt beobachtbar. Für uns als Beobachter*in 2. Ordnung erschien das System wie folgt:

- Es beobachtete seine Umwelt (die Software);
- es nahm dabei offensichtlich Unterscheidungen vor: Aus den vielen Möglichkeiten, wahrgenommene Reize als relevant einzustufen, wählte das System im zeitlichen Verlauf jeweils eine und schloss damit andere Möglichkeiten aus;
- dadurch wurden bestimmte Anschluss-Unterscheidungen wahrscheinlicher als andere.

Erkennen konnten wir als Beobachter*in 2. Ordnung dies an den sichtbaren Aktivitäten des Systems. Eine konkrete Eingabe auf der „Benutzeroberfläche"-Seite der **Software**-Form (ein Mausklick, ein Scrollen, ein Tastendruck usw.) deutet auf eine Unterscheidung hin, die das System vorher hinsichtlich der „Funktion"-Seite der **Software**-Form vor dem Hintergrund der „Plan"-Seite der **Ziel**-Form getroffen haben muss. Die Eingabe zeigt uns außerdem, dass sich das System auf der „Reproduktion"-Seite der **System**-Form ‚befindet'.

In Alltagssprache: Die Nutzerin sah eine Reihe von Optionen und wählte hinsichtlich ihres Nutzungsziels eine aus, die ihr als am passendsten erschien. Die Nutzerin nahm fortlaufend Eingaben vor und verarbeitete Ausgaben in einer Weise, die zu weiteren Eingaben führten. Die Software als Umwelt des Systems erbrachte damit offensichtlich Leistungen, die in der Situation für das System relevant erschienen. Neben konkreten Eingaben verweisen auch Äußerungen der Nutzerin auf diese zugrunde liegende Unterscheidungen.

Doch plötzlich kommt es zu einer Irritation – es wird nicht mehr in erwarteter Weise auf eine Eingabe (Zeile 1, „ich geh auf Mails versenden") reagiert (Zeile 2, „da passiert nichts weiter"). An dieser Stelle kommt es zu dem, was ich in Donick (2016) und in vorliegendem Buch als *Bruch* der Kopplung von System und Umwelt bezeichne. War bis dahin die „Reproduktion"-Seite der **System**-Form zu beobachten, steht nun die „Störung"-Seite der **System**-Form im Fokus (Zeile 2, „seltsam") – die Nutzung geht nicht weiter, weil auf den Mausklick keine sichtbare Funktion folgt. Die produzierten kommunikativen Beiträge verweisen auf die „Abweichung"-Seite der **Ziel**-Form und drücken die enttäuschte Erwartung an die „Funktion"-Seite der **Software**-Form aus: Die Erwartung war natürlich, dass nach dem Klicken auf „Mails versenden" ein Versenden von Mails erfolgt und dass eine Rückmeldung der Software darüber Auskunft gibt. Die Enttäuschung war, dass dies nicht geschah.

Obwohl diese Äußerungen nicht zur Nutzung der Software beitragen oder direkt Nutzungsaktivitäten kommentieren, dienen sie dem System gleichwohl zur Stabilisierung (denn jede Außenseite einer Zweiseiten-Form impliziert die Möglichkeit der Rückkehr zur Innenseite). Mit der ironischen Äußerung „das is n bisschen lustig" (Zeile 3) distanziert sich das System von der Dringlichkeit der Störung, was ihm Zeit verschafft, eine nächste brauchbare Unterscheidung zu finden, mit der es die Nutzung fortsetzen kann. Zeit ist in der Systemtheorie ein wichtiger Faktor für den Selbsterhalt eines Systems: Jede Unterscheidung erfordert Zeit, und Operationen, die dem System mehr Zeit verschaffen (wie ironische Distanzierung und Vertrauen) oder die für Unterscheidungen benötigte Zeit reduzieren (wie Zwang), können hilfreich sein (vgl. Luhmann 2009, S. 74 f.).

Tatsächlich findet das System im beobachteten Beispiel nach der Distanzierung die nötige Unterscheidung schnell: „weil ich hab ja keine E-Mail-Adresse von diesen Leuten" (Zeile 4). Ohne E-Mail-Adressen kann natürlich das Klicken auf einen Senden-Button (Zeile 1) nicht erfolgreich sein. Nach dieser Unterscheidung waren nun Schritte möglich, die einen Wechsel zurück auf die Nutzung-Seite der Form erlaubten (nämlich, die Mail-Adressen zu besorgen und zu ergänzen).

Aus praktischer Sicht erhalten wir an der Stelle einen Hinweis auf eine nicht-relevante Kommunikationsweise der getesteten Software. Von außen tat die Software scheinbar gar

nichts. Ich sage „scheinbar“, weil die Software nach Klick auf den Button durchaus etwas tat – sie prüfte, ob gültige Empfänger-Adressen eingetragen waren. Diese Aktivität war für die Situation auch sehr relevant – aber sie geschah unsichtbar, das heißt, die Software machte ihre Relevanz nicht deutlich. An der Stelle wäre natürlich eine Meldung sinnvoll gewesen. Bei ausreichender Relevanz der Meldung für die Situation (etwa: ‚Bitte tragen Sie mindestens eine E-Mail-Adresse in das Empfänger-Feld ein.‘) wäre es vermutlich gar nicht zur Unterbrechung der Nutzung bzw. einem Bruch der Kopplung gekommen.

3.3 Wozu brauchen wir das?

Sie haben in diesem Kapitel drei Formen kennengelernt, die Ihnen helfen, die Beziehung von Nutzer*innen und Software zu beschreiben:

- **Nutzer*in** als System-Form, die zwischen Reproduktion und Störung wechseln;
- **Software** als Design-Form, die vom Verhältnis von Funktion und Form bestimmt ist;
- **Nutzung** als Ziel-Form, die mit der Enttäuschung von Erwartungen rechnet.

Tab. 3.1 Beziehungen der Zweiseiten-Formen der Softwarenutzung (nach Donick 2016 und Baecker 2007) zueinander, vor dem Hintergrund von Relevanz

	Design: Funktion/Form *Software: Funktion/ Oberfläche*	**Erwartung: Erwartung/Enttäuschung** *Nutzung: Ziel/Abweichung*
System: Reproduktion/ Störung *Nutzer*in: Nutzung/Störung*	Über beobachtbare Formen baut das Nutzer*in-System eine Beziehung zu seiner Software-Umwelt auf. Nur über die Formen kann es auf die hinter den Formen liegende Funktionalität verfügen. Gelingt dies, kann man der Software Relevanz unterstellen; gelingt dies nicht, ist die Reproduktion des Nutzer*in-Systems gefährdet (Nutzung bricht ggf. ab.)	Menschen sind als Nutzer*in-System beobachtbar, wenn sie ein bestimmtes Ziel haben bzw. einen Plan verfolgen. Das Ziel ist Hintergrund der Nutzung bzw. der Reproduktion des Systems. Abweichungen vom Ziel stören die Reproduktion des Nutzer*in-Systems. Es muss die Abweichung in relevanter Weise an den Plan/das Ziel anknüpfen, um sich weiter reproduzieren können.
Design: Funktion/Form *Software: Funktion/ Oberfläche*		Nutzer*innen haben bestimmte Erwartungen an Funktion und Form von Software. Design wirkt umso relevanter, je besser es zu den Erwartungen passt. Abweichungen können Relevanz in Frage stellen, sie können sich aber auch im Gegenteil als relevanter herausstellen, wenn dadurch deutlich wird, dass die Erwartungen selbst vor anderen Hintergründen ungeeignet waren.

Diese drei Formen sind ausreichend, um die Nutzung von Software in Hinblick auf Relevanzprozesse zu beschreiben. Sie bilden einen Hintergrund, vor dem Sie die folgenden Kapitel besser einordnen können. In der Praxis müssen Sie sich Einzelfälle anschauen, um mit Hilfe der Formen Aussagen über Ihre eigenen Produkte treffen zu können, aber Sie werden die Beobachtungen immer an diese Unterscheidungen anbinden können. In Tab. 3.1 habe ich die Formen einmal vor dem Hintergrund von Relevanz in Beziehung gesetzt.

Literatur

http://www.agilemodeling.com/artifacts/acceptanceTests.htm

Baecker, Dirk: Form und Formen der Kommunikation. Frankfurt/Main 2007.

Baraldi, Claudio / Corsi, Giancardo / Esposito, Elena: GLU. Glossar zu Niklas Luhmanns Theorie sozialer Systeme. Frankfurt/Main 1998.

Donick, Mario: „Offensichtlich weigert sich Facebook, mir darauf eine Antwort zu geben“: Strukturelle Analysen und sinnfunktionale Interpretationen zu Unsicherheit und Ordnung der Computernutzung. Hamburg 2016.

Donick, Mario: Die Unschuld der Maschinen. Technikvertrauen in einer smarten Welt. Wiesbaden 2019.

Grice, Paul: Logic and Conversation. In: Ders. (Hrsg.): Studies in the Way of Words. Cambridge/London 1991, S. 22–40.

Linke, Angelika / Nussbaumer, Markus / Portmann, Paul R.: Studienbuch Linguistik. Tübingen 2004.

Luhmann, Niklas: Soziale Systeme. Frankfurt/Main 1996.

Luhmann, Niklas: Vertrauen. Stuttgart 2009.

Maturana, Humberto R.: Biologie und Realität. Frankfurt/Main 2000.

Sperber, Dan / Wilson, Deidre: Relevance. Communication and Cognition. Oxford 1995.

Suchman, Lucy: Human-Machine Reconfigurations. Plans and Situated Actions 2nd Edition. Cambridge 2007.

Sweller, John: Cognitive Load During Problem Solving: Effects on Learning. In: Cognitive Science 12, 1988, S. 257–285.

Teil II

Anwendung – Wie wir Softwarenutzung beobachten können

Softwarenutzung strukturiert beobachten

4

Inhaltsverzeichnis

4.1 Was heißt „Softwarenutzung beobachten“?

Das Verfahren, dass Sie ab diesem Kapitel kennenlernen, gehört zu den qualitativen und interpretativen Beobachtungsverfahren (vgl. Koller 2008). Beobachtung ist wörtlich zu verstehen: Beispielsweise filmen Sie Menschen dabei, wie diese Ihre Software benutzen, Sie zeichnen sprachliche Äußerungen dieser Nutzer*innen auf (sogenanntes ‚Lautes Denken‘, vgl. van Someron et al. 1994), Sie protokollieren Softwareaktivität usw. Sie erheben also eine Menge Daten, mit denen Sie hinterher in der Lage sind, die relevanten Parameter der Situation zu rekonstruieren. Qualitative Verfahren eignen sich, um in die Tiefe des beobachteten Gegenstands – oft einer Kommunikationssituation – zu dringen und herauszuarbeiten, warum bestimmte Situationen gelingen und andere weniger. Wie Sie dies tun können und was es dabei zu beachten gibt, schauen wir uns im Rest dieses Buches an. Für einen Überblick ist der vollständige Ablauf des Verfahrens in Abb. 4.1 dargestellt.

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8_4

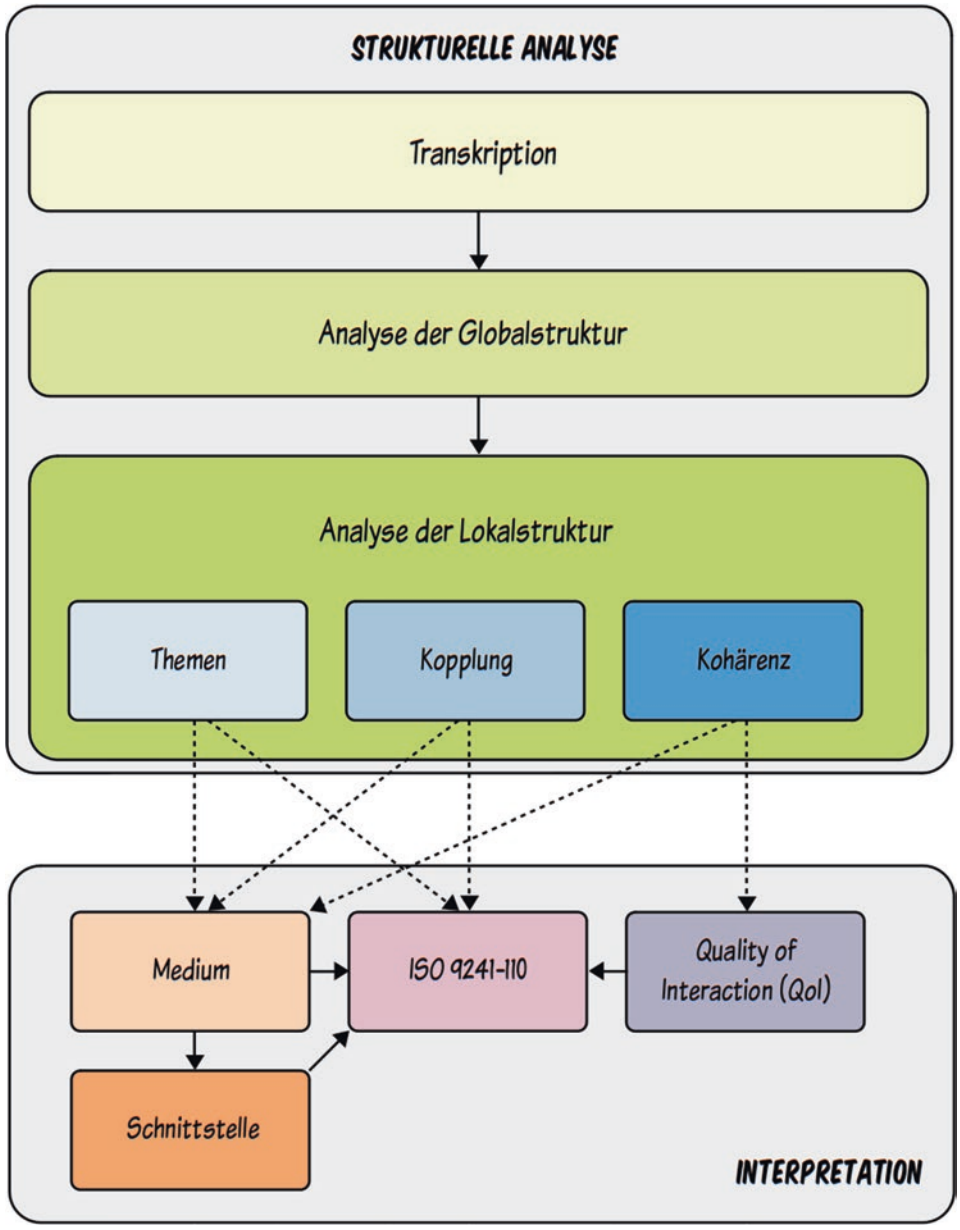

Abb. 4.1 Überblick über das vollständige Beobachtungsverfahren

Um den Nutzen qualitativer Beobachtung zu verdeutlichen, denken Sie zuvor noch einmal zurück an die vorigen drei Kapitel dieses Buches, in denen ich auf die Systemtheorie Niklas Luhmanns und Dirk Baeckers verwiesen habe. Darin wurde immer wieder betont, dass Menschen als Systeme beschrieben werden können, die sich gegenseitig beobachten.

Systeme beobachten andere Systeme, und sie können auch beobachten, dass ein anderes System beobachtet (Beobachter 2. Ordnung), aber Systeme können nicht in andere Systeme hineinschauen. Das ist ein Grundgedanke systemtheoretisch orientierter Kommunikationsforschung.

Ein nichtsdestotrotz nötiger Austausch zwischen Systemen ist in Form von Leistungsbeziehungen (strukturellen Kopplungen) möglich. Luhmann selbst hat die Beziehung von Mensch und Technik in dieser Form nicht beschrieben, es ist aber möglich und sinnvoll, dies zu tun (vgl. Donick 2016). Ob zwischen einer Software und deren Nutzer*in eine Leistungsbeziehung besteht (die Software also in der Situation für die entsprechende Person gut funktioniert), kann nur durch Beobachtung während der Nutzung ermittelt werden.

Quantitative und qualitative Forschungsmethoden

Wird Forschung an *naturwissenschaftlichen* Methoden ausgerichtet (wie es zum Beispiel Psychologie und Sozialpsychologie tun), dann entscheidet man sich meist für *quantitative Methoden*. Bei diesen steht der in Zahlen ausgedrückte Ausprägungsgrad von Merkmalen im Vordergrund (Klauer 2006, S. 84), ohne aber diese Merkmale im Detail tief gehend zu betrachten. Es sollen möglichst allgemeingültige Wenn-dann-Aussagen generiert werden, die etwas erklären, eine Prognose für die Zukunft erlauben oder handlungsleitend sind (ebd., S. 40–45). Typische Beispiele für solche Verfahren sind Experimente, Quasi-Experimente (ebd., S. 77f.) oder Korrelationsstudien (ebd., S. 81). Die Fallzahlen untersuchter Proband*innen können sehr hoch sein, sodass statistisch signifikante Aussagen möglich sind – zum Beispiel, dass ein hoher Prozentsatz von Test-Nutzer*innen mit der neuen Benutzerführung einer Software signifikant besser zurechtkommt als mit der alten Oberfläche, was dann dahingehend handlungsleitend sein kann, die neue Oberfläche für alle Nutzer*innen auszurollen. Erkenntnistheoretisch liegt quantitativen Methoden das Ideal der Objektivität zugrunde, also die Überzeugung, dass es grundsätzlich möglich ist, Aussagen über die Welt zu machen, die von allen Menschen geteilt werden können.

Bei der *qualitativen Forschung* identifizieren Sie Merkmale, die für Ihren Untersuchungsgegenstand typisch sind, und Sie bestimmen diese Merkmale näher (Klauer 2006, S. 84). Sie werden aber kaum in der Lage sein, aus diesen Merkmalen allgemeingültige Erklärungen, Prognosen oder Handlungsanweisungen abzuleiten. Beispielsweise werden in der Kommunikationswissenschaft gesprächsanalytische Methoden genutzt, um ein einzelnes Gespräch zwischen zwei oder mehr Akteuren im Detail zu rekonstruieren und zu interpretieren. So kann gezeigt werden, welche interaktiven Praktiken *diese* beobachteten Akteure anwenden, um gemeinsam ihre soziale Wirklichkeit zu konstruieren. Dies lässt sich aber nicht ohne weiteres auf alle Menschen verallgemeinern. Bei der qualitativen Beobachtung von Softwarenutzung können Sie durch die Beobachtung von Nutzer*innen zwar im Detail erfahren, wie sich diese die Software erschließen, und daraus entscheidende Hinweise für die weitere Entwicklung der Software erhalten, aber Sie können aufgrund der meist geringen Fallzahlen (vgl. Abschn. 4.3) daraus nicht schließen, dass die beobachteten Merkmale typisch für die meisten Nutzer*innen wären.

Softwarenutzung zu beobachten, um praktisch verwertbare Ergebnisse für die Softwareentwicklung zu erhalten, heißt also, unsere normalerweise unbewussten Beobachtungsvorgänge der Alltagskommunikation zu reflektieren und zielgerichtet als Werkzeug einzusetzen:

> Das Ziel der Beobachtung ist, den Aufbau, das Bestehen oder die Beendigung einer Leistungsbeziehung von Nutzer*in und Software zu rekonstruieren und daraus Schlüsse für die Entwicklung oder Verbesserung der jeweiligen Software abzuleiten.

Freilich sind solche Beobachtungen mit einem gewissen Aufwand verbunden, der auch nicht durch im Nachhinein geführte Interviews, Fragebögen oder gar ‚Mini-Befragungen', die manchmal nur aus einer Frage mit zwei Antwortmöglichkeiten bestehen, ersetzt werden kann (aber im Sinne einer Methodentriangulation sehr wohl ergänzt).

Das hier vorgestellte Verfahren ist nicht nur qualitativ, sondern interpretativ. Manchmal werden beide Begriffe synonym verwendet (Koller 2008, S. 606), aber es ist recht nützlich, sie als Paar zu nutzen. Dadurch wird betont, dass die Auswertung qualitativer Daten auf jeden Fall eine subjektive Interpretationsleistung ist, die keinen objektiven Kriterien genügen kann. Schon die Transkription (Verschriftlichung) eines Protokolls ‚Lauten Denkens' aus einer Beobachtungssitzung kann eine Interpretationsleistung sein, wenn im Transkript auch nicht-sprachliche Merkmale markiert werden sollen (wie etwa Emotionalität). Und die anschließende Auswertung des Transkripts hinsichtlich bestimmter Fragestellungen bezieht noch stärker die individuelle Weltsicht der auswertenden Person ein. Um eine gewisse Zuverlässigkeit zu erreichen, lässt man daher manchmal mehrere Personen dieselben Daten unabhängig voneinander auswerten und identifiziert dann Unterschiede und Gemeinsamkeiten der Interpretationen (vgl. Kasten „Gütekriterien").

Gütekriterien

Bei quantitativer Forschung, die sich am naturwissenschaftlichen Experiment orientiert (vgl. den Kasten „Quantitative und qualitative Forschungsmethoden") gibt es klare Gütekriterien, die die Qualität von Forschungsprozess und Ergebnissen beurteilen helfen. *Objektivität* meint die Vermeidung subjektiver Einflüsse (Jürgens 2005, S. 74f.). Dies betrifft die Durchführung der Untersuchung, die Auswertung und die Interpretation der erhobenen Daten (ebd.). *Reliabilität* bedeutet, dass die Erhebung der Daten exakt geschieht und Messfehler so gering wie möglich ausfallen (ebd., S. 76). Die Kriterien Objektivität und Reliabilität haben *Validität* zum Ziel (ebd., S. 77), das heißt, dass tatsächlich das Merkmal erhoben wird, um das es geht, und nicht ein anderes (ebd.).

Qualitative Forschung ist mehr am Einzelfall interessiert als an objektiven, verallgemeinerbaren Aussagen. Daher lassen sich die genannten Kriterien oft nicht vollständig anwenden. Stattdessen stehen die *intersubjektive Nachvollziehbarkeit* des Forschungsprozesses, dessen *Gegenstandsangemessenheit* und *Reflexion* im Vordergrund (Koller 2008, S. 620). Ein weiteres Kriterium kann es sein, wenn sich aus den Einzelfällen

bestimmte *Typen* ableiten lassen, auch wenn diese quantitativ nicht bestimmbar sind (ebd., S. 619).

Wegen der ‚weicheren' Gütekriterien werden qualitative Methoden oft als zwar nützliche Ergänzung zu quantitativen Methoden betrachtet, aber nicht als Ersatz für diese akzeptiert. Diese Einstellung besteht insbesondere in den eher positivistischen Naturwissenschaften und der Psychologie. So ist etwa Klauer (2006) der Ansicht, dass sich qualitative Methoden zur Generierung von Hypothesen eignen würden, dass diese Hypothesen aber in einem weiteren Schritt mit quantitativen Methoden überprüft werden müssten (Klauer 2006, S. 85).

Diese Sicht kann so weit gehen, qualitativen Methoden insgesamt abzusprechen, Forschung zu sein (ebd., S. 84). Solcherart Dogmatismus übersieht jedoch, dass nicht alle interessierenden Probleme – auch viele, die praktischer Art sind – als Fragestellung klassisch-kausaler Art (vgl. Kasten „Die Frage bestimmt die Antwort" in Abschn. 4.2) formuliert und quantitativ untersucht werden können.

4.2 Geeignete Fragestellungen, benötigte Daten

Beobachtung ist ein Mittel zum Zweck. Damit man nur so geeignete Daten erhebt und nur so viel wie nötig (aber so wenig wie möglich), ist es wichtig, dass Sie sich im Vorfeld über diesen Zweck klar werden. Wenn Sie den Zweck kennen, können Sie Hypothesen und Fragestellungen formulieren, anhand derer Sie die Beobachtung einrichten. Fragen Sie sich also als erstes:

- *Was will ich* wissen*?* Das heißt etwa: Welche Fakten wollen Sie sammeln? Wofür suchen Sie Erklärungen? Welche Alternativen wollen Sie aufdecken?
- *Was will ich* erreichen*?* Das heißt etwa: Was wollen Sie optimieren? Was soll mit Hilfe der Ergebnisse hinterher anders gemacht werden als vorher?

Um Zeit und Geld nicht unnötig zu vergeuden, ist es wirklich wichtig, dass Sie sich über die Antworten auf diese Fragen klar werden. Sie könnten die beiden Fragen auch zusammenfassen: „Wozu brauche ich die Beobachtungen?" Und tatsächlich werden Sie manchmal feststellen, dass Sie sie gar nicht brauchen. Es gibt Situationen, in denen qualitative Beobachtungen ungeeignet sind – dummerweise ist das Risiko jeder Forschung, dass man erst hinterher (nach beendeter Untersuchung) weiß, ob sie sinnvoll war (aber auch das wäre dann ein Ergebnis, aus dem sich oft Schlüsse ziehen lassen.)

Denken Sie bitte an Ihre eigene Arbeit und Erfahrungen mit Softwareentwicklung. Welche möglichen Zielstellungen für die Beobachtung von Softwarenutzung fallen Ihnen da ein? Wofür könnten solche Beobachtungen für Sie nützlich sein?

Hier sind einige Beispiele (vgl. auch den Kasten „Die Frage bestimmt die Antwort“):

- Wollen Sie nur einen allgemeinen Eindruck davon gewinnen, wie Ihr Softwareprodukt während einer bestimmten Entwicklungsphase aufgenommen wird?
- Möchten Sie verschiedene Ansätze für Form und Funktion vergleichen, um herauszufinden, welcher besser geeignet ist, bevor Sie sich für einen entscheiden?
- Oder wollen Sie einen Ansatz testen und erhoffen sich dadurch Aufschluss über mögliche Alternativen, an die Sie selbst noch gar nicht gedacht haben?
- Fragen Sie sich, was an einem bereits etablierten Produkt in der nächsten Version besser gemacht werden kann?
- Interessiert es Sie, ob Ihr Produkt die Erwartungen der Nutzer*innen erfüllt?

Diese und ähnliche Fragen lassen sich natürlich auch schneller und kostengünstiger über Fragebögen und Interviews beantworten, oft helfen auch automatisierte Akzeptanztests. Beobachtungen haben jedoch den Vorteil, dass Ihnen dabei Zusammenhänge der Nutzung bewusst werden können, die den Nutzer*innen beim Beantworten von Fragen oder beim Durchführen von Akzeptanztests gar nicht auffallen würden. Durch Beobachtung erhalten Sie einen direkteren Zugriff auf tatsächliches Handeln in der Situation, und Sie wissen durch dieses Buch ja, warum das wichtig und nützlich ist.

Die Frage bestimmt die Antwort

So wie sich qualitative Verfahren insgesamt von quantitativen Verfahren in der Zielsetzung unterscheiden, so hängt auch die Entscheidung für ein bestimmtes qualitatives Verfahren von der Fragestellung ab (vgl. Donick 2016, S. 147–159):

Bei einem *klassischen kausalen Ansatz* (wie in Naturwissenschaft und Psychologie üblich) sind Sie sehr häufig an Erklärungen für ein Phänomen interessiert – Sie wollen wissen, warum etwas der Fall ist, statt die Beobachtung nur zu beschreiben. In der Tradition der Wissenschaftstheoretiker Karl Popper (1935) sowie Carl G. Hempel und Paul Oppenheim (1948) ist es für kausale Erklärungen nur erlaubt, vom Allgemeinen auf das Spezielle zu schließen (Deduktion, vgl. Engemeier et al. 2011,167), aber nicht umgekehrt (Induktion, ebd.). Der Ansatz generiert Aussagen der Form „wenn *p*, dann (typischerweise) *q*“. Wenn beispielsweise eine Software auf bestimmte Weise gestaltet ist, dann tun Nutzer*innen dieses oder jenes. Grundsätzlich ist es auch mit qualitativen Verfahren möglich, kausale Erklärungen zu liefern, allerdings steht dem in der Praxis die – auch wegen des Aufwands – meist sehr geringe Fallzahl von Proband*innen im Weg. Am ehesten eignen sich noch Verfahren, die basierend auf der Auswertung von Äußerungen, „Dokumenten und ähnlichem“ eine Kategorienbildung vornehmen, um hypothesengeleitet Schlüsse zu ziehen, wie etwa qualitative Inhaltsanalyse nach Mayring (2002).

Bei *funktionalen Ansätzen* nach Luhmanns Systemtheorie setzen Sie beobachtete Einzelfälle in Differenz zu möglichen Alternativen. In einer sogenannten funktionalen Analyse erklären Sie nicht, was warum ist, sondern zeigen, was statt des Beobachteten

möglich wäre. Das nennt man Funktionsäquivalent. Solche Ansätze generieren Aussagen der Form „*q* ist hier, weil *p*, aber deshalb könnte statt *q* auch *r*, *s*, *t*, … sein“. Wenn Nutzer*innen etwa einen bestimmten Bedienweg beschreiten, zeigt sich vielleicht, dass ein alternativer, aber in der Software nicht vorgesehener Bedienweg sinnvoller gewesen wäre. Beispiele für systemtheoretisch-funktionale Ansätze sind die Differenztheoretische Textanalyse (vgl. Ammer 2008; Huth 2008) und die systemtheoretische Anwendung der Objektiven Hermeneutik (Sutter 2010).

Bei *Ansätzen in der Tradition der Ethnomethodologie* schauen Sie sich Äußerungen von Menschen im zeitlichen Verlauf an, um nachzuvollziehen, mit welchen Praktiken diese Menschen ihre soziale Welt konstruieren (vgl. Bergmann 1994), zum Beispiel Normen, Regeln und Pläne. Solche Ansätze generieren Aussagen der Form „*q* ist (hier im Einzelfall), weil *p*“. Beispielsweise könnten Sie feststellen, dass ein Bedienfehler vorliegt, weil die Person vom vorgesehenen Plan abgewichen ist. Sie erkennen dann, dass Pläne erst in der konkreten Situation handlungsrelevant werden, indem Menschen sie beobachten und davon berichten („observable-reportable“, Hill und Crittenden 1968). Das nennt man „accounts“, vom Verb „to be accountable for something“ (für etwas verantwortlich sein) und vom Adjektiv „accountable“ (erklärlich). Accounts erklären die Herstellung von Ordnung und sind gleichzeitig am Entstehen der Ordnung beteiligt. Begründer der Ethnomethodologie war Harold Garfinkel; für die Techniksoziologie besonders prägend war die Arbeit von Lucy Suchman (2007).

Bei der *Kohärenzanalyse* nach Donick (2016) werden Aspekte der drei anderen genannten Ansätze verbunden. Sie arbeiten heraus, wie einzelne Nutzer*innen schrittweise die Komplexität der genutzten Software einschränken, indem sie verschiedene Alternativen testen. Dieser Ansatz zeigt, wie Nutzer*innen sich für eine Alternative entscheiden und andere ausschließen. Als konkrete Evaluationsmethodik wurde der Ansatz in Donick und Tavangarian (2010) gezeigt, dann in Donick et al. (2012) mit dem Konzept der Quality of Interaction (QoI) verbunden (vgl. Kap. 6) und schließlich in Donick (2016) detailliert ausgearbeitet und empirisch untersucht. Anschließend wurde der Ansatz in verschiedenen Usability-Studien in der Praxis getestet.

Wenn Sie sich über Ihre Ziele und Fragestellung im Klaren sind, sollten Sie einige Annahmen über erwartete (oder nicht erwartete) Ergebnisse formulieren, um „das Eingeständnis der Komplexitätsunterlegenheit“ (Luhmann 1992, S. 370) des forschenden Systems im Vergleich zur komplexeren Umwelt „auf[zu]fangen“ (ebd.) – also der Tatsache Rechnung zu tragen, dass die Beobachter*innen immer weniger komplex sind als die beobachtete Umwelt, und dass daher jede Untersuchung – jede Entscheidung für eine Fragestellung und eine Beobachtungsmethode – auch scheitern kann (ebd., S. 254–255). Die Formulierung von Annahmen hilft, mit dieser Ungewissheit umzugehen (vgl. Donick 2016, S. 115 f.).

Wenn Sie Ihre Annahmen formalisieren, dann bilden Sie *Hypothesen*. Sie haben dann zum Beispiel vier forschungsleitende Hypothesen, die Sie als Aussagesätze formulieren und im Verlauf der Untersuchung widerlegen (falsifizieren) oder bestätigen (verifizieren).

Hypothesen sind Aussagen, die Vermutungen über die (als objektiv angenommene) Wirklichkeit ausdrücken – über ihre Eigenschaften oder über ihre kausalen Zusammenhänge:

- In einer *positiven Existenzhypothese* behaupten Sie, dass es etwas gibt, beispielsweise: „Mit der neuen Version der Software können Nutzer*innen schneller arbeiten als mit der alten Version“. In dieser Hypothese behaupten Sie das Vorhandensein einer Eigenschaft, die Sie nur verifizieren (bestätigen), aber nicht falsifizieren (widerlegen) können (Krapp und Weidenmann 2006, S. 736). Denn auch wenn Sie in Ihrer Untersuchung nicht bestätigen können, dass Ihre Existenzbehauptung zutrifft, könnte es das Behauptete ja trotzdem irgendwo auf der Welt geben, und diese Möglichkeit können Sie nicht letztgültig widerlegen.
- In einer *negativen Existenzhypothese* behaupten Sie, dass es etwas nicht gibt, beispielsweise: „Es gibt keine Nutzer*innen, die mit der neuen Software Schwierigkeiten haben.“ Diese Hypothese können Sie nur falsifizieren, aber nicht verifizieren (ebd.). Denn auch wenn die Beispielhypothese auf alle Ihre Proband*innen zuträfe, könnte es ja irgendwo auf der Welt eine Person geben, bei der sie nicht zutrifft, und das können Sie nicht letztgültig bestätigen.
- In einer *Gesetzeshypothese* behaupten Sie einen Wenn-dann-Zusammenhang zwischen zwei Aspekten der Wirklichkeit, beispielsweise: „Wenn mehr Icons in die Symbolleiste passen, können Nutzer*innen ihre Aufgaben schneller erledigen.“ Solche Gesetzeshypothesen können Sie nur falsifizieren, aber nicht verifizieren (ebd.). Denn auch wenn die Beispielhypothese auf alle Ihre Proband*innen zuträfe, könnte es ja irgendwo auf der Welt eine Person geben, bei der dieser Zusammenhang nicht erkennbar ist, und das können Sie nicht letztgültig bestätigen.

In gewisser Weise mit Hypothesen verwandt sind sogenannte *sensibilisierende Konzepte*. Der Begriff stammt aus der Grounded Theory (vgl. Strübing 2008; Alheit 1999). Sensibilisierende Konzepte entstammen Ihrer Lebenserfahrung und Berufspraxis, oder entstehen theoriegeleitet. Sie dienen einer ersten Orientierung und anstatt sie zu widerlegen oder zu bestätigen, verfeinert man sie eher im Laufe einer Untersuchung. Im Gegensatz zu den klar definierten Existenz- und Gesetzeshypothesen, die Sie verifizieren oder falsifizieren können, sind sensibilisierende Konzepte viel offener und damit auch formal nicht an die Aussageform gebunden. Sie zeugen von einem fortschreitenden Lernprozess (Alheit 1999, S. 10), der im Laufe der Sensibilisierung der Forscher*innen für das Forschungsobjekt zu beobachten ist.

Das ‚Konzept‘ ist kein Dokument im Sinne einer Ausformulierung von Annahmen, sondern zunächst einmal Ihre kognitive Vorstellung zu einem Thema, die Sie schrittweise verfeinern, während Sie sich für das Thema sensibilisieren. Sie sensibilisieren sich für ein Thema, indem Sie Alltagsbeobachtungen machen, etwas zum Thema lesen, mit anderen Menschen sprechen usw. Im Verlauf werden Sie Ihr kognitives Konzept natürlich trotzdem in für Sie und Ihr Forschungsinteresse geeigneter Weise festhalten.

Beispiel für sensibilisierende Konzepte

Ein Beispiel für eine systemtheoretische Anwendung sensibilisierender Konzepte gebe ich in meinen Untersuchungen zu softwarebezogener Ungewissheit (Donick 2016, S. 115–146). In der Untersuchung habe ich rekonstruiert, wie Nutzer*innen mit Problemen bei der Computernutzung umgehen – wie sie versuchen, die Nutzung trotz Problemen fortzusetzen oder Alternativen zu finden. Bevor ich die eigentlichen Beobachtungen durchgeführt habe, habe ich sensibilisierende Konzepte gebildet.

Im ersten Schritt habe ich meine eigenen Alltagserfahrungen mit Computernutzung reflektiert (ebd., S. 116). Ich habe über Situationen nachgedacht, in denen ich selbst Schwierigkeiten mit Software hatte und habe mich erinnert, wie ich damit umgegangen bin.

Im zweiten Schritt habe ich meine Reflexionen abgeglichen mit ersten kurzen, sondierenden Beobachtungen anderer Personen. Noch sehr informell habe ich mir also angeschaut, wie andere Menschen mit Schwierigkeiten umgehen. Dadurch verfeinerte ich die sensibilisierenden Konzepte aus Schritt 1.

Im dritten Schritt habe ich die bis dahin gemachten Selbst- und Fremdbeobachtungen mit theoretischen Hintergründen aus der Literatur verknüpft (ebd., S. 117). Damit verließ ich die Ebene der Alltagsbeobachtungen und verfeinerte die sensibilisierenden Konzepte aus Schritt 1 und 2 anhand vorhandener wissenschaftlicher Forschung.

Im letzten Schritt formalisierte ich meine bis dahin vorhandenen Konzepte – das heißt, ich schrieb sie auf. Aber dazu bildete ich keine Hypothesen, sondern ich drückte jedes Konzept als sogenannte *form* aus (ebd.; im Sinne der Laws of Form nach Spencer-Brown 2011; vgl. Baecker 2007 und Kap. 2 dieses Buches). Anstatt als Behauptung über die Wirklichkeit konnte man jede *form* als Struktur- und Prozessmodell verstehen, bei dem gerade auch die Möglichkeit seiner Nichtgeltung und damit die Offenheit für Alternativen bzw. Funktionsäquivalente (vgl. Kasten „Die Frage bestimmt die Antwort“) mitgedacht war.

Die so gebildeten Formen konnte ich in meiner Untersuchung nicht verifizieren oder falsifizieren, und dafür waren sie auch nicht gedacht. Sie bildeten einen Hintergrund, von dem meine Detailanalysen von Beobachtungen menschlichen Handelns strukturiert und vor dem sie intersubjektiv nachvollziehbar wurden.

Für qualitative interpretative Verfahren sind sensibilisierende Konzepte oft besser geeignet als formal strengere Hypothesen. Doch ob Sie lieber mit Hypothesen oder lieber mit sensibilisierenden Konzepten arbeiten, bleibt Ihnen überlassen; es hängt wiederum von Ihrem Ziel und Ihren Fragen ab, was sich besser eignet. Zudem ist es möglich, beides zu kombinieren – sie können mit sensibilisierenden Konzepten beginnen, darauf aufbauend ihre qualitative Beobachtung anlegen und daraus dann formalere Hypothesen gewinnen, die Sie vielleicht sogar in einer größer angelegten quantitativen Untersuchung überprüfen.

Vor einer Untersuchung Annahmen zu formulieren (ob nun Hypothesen oder sensibilisierende Konzepte) verlangt, sich schon vor der Beobachtung mit dem Gegenstand reflektiert auseinanderzusetzen. Das kann ganze Vorstudien vor der eigentlichen Untersuchung zur Folge haben. Manchmal ist so etwas aber nicht erwünscht – und zwar nicht bloß, weil das noch mehr Zeit und Geld kostet, sondern weil man ganz bewusst ergebnisoffen beobachten möchte, ohne durch Vorannahmen beeinflusst zu werden. Wenn das in Ihrem Fall auch so ist – dann machen Sie das.

Überhaupt gilt, dass in diesem Buch und in anderen Büchern zu Forschungsmethoden zwar viele Möglichkeiten vorgestellt und Beispiele geliefert werden, dass aber die jeweils für Sie geeignete Kombination von Methoden und deren Umsetzung nur durch Sie selbst beurteilt werden kann. Sie tun das anhand Ihrer Erfahrung, durch Ausprobieren oder in der Kommunikation mit anderen Menschen. Der Philosoph Paul Feyerabend schrieb in seinem Buch „Wider den Methodenzwang", dass „die Schachzüge, die [das Forschen] fördern, […] dem Forscher oft erst nach Vollendung der Forschung klar [werden]" (Feyerabend 1986, S. 376). Dies ist ein Rat, den man stets beherzigen sollte.

4.3 Beobachtungen planen und organisieren

Sie haben sich über den Zweck Ihrer Beobachtung Gedanken gemacht, eine Fragestellung gefunden und möglicherweise auch Hypothesen formuliert oder sensibilisierende Konzepte gebildet. Nun brennen Sie darauf, loszulegen und Antworten auf Ihre Fragen zu erhalten. Ganz so schnell geht es dann aber doch nicht weiter, denn um Nutzer*innen zu beobachten – brauchen Sie erst mal Nutzer*innen, die geeignet und dazu bereit sind (das wären Ihre Proband*innen – auch wenn Sie sie nicht im Labor beobachten, sondern z. B. in deren Büro per Screen- und Audiorecorder). Auch Ort und Zeit der Beobachtung müssen geklärt sein, ebenso die Dauer der Beobachtung, und möglicherweise eine Aufwandsentschädigung für die Proband*innen.

4.3.1 Die Suche nach Proband*innen

Wie alles, hängt auch die Suche nach Proband*innen davon ab, was Sie wissen wollen. Es kommt nicht nur darauf an, überhaupt Menschen zu finden, die zur Teilnahme bereit sind, sondern es sollten Personen sein, die repräsentativ für aktuelle oder künftige Nutzer*innen sind. Folgende Parameter können je nach Projekt und Fragestellung von Bedeutung sein:

- *Tätigkeit:* Dies kann beeinflussen, welche Vorerfahrungen die Personen in dem Anwendungsbereich Ihrer Software schon mitbringen bzw. welche Erwartungen sie an die zu testende Software haben.
- *Alter:* Dies ist mitunter (aber nicht immer) ein Faktor für die Erfahrung mit bestimmten Arten von Hard- und Software. Es ist ein Fakt, dass es uns mit zunehmenden Lebensjahren schwerer fällt, uns an Neuerungen zu gewöhnen.

- *Vorerfahrung als Studienteilnehmer*in:* Personen, die bereits an anderen Tests teilgenommen haben, werden wahrscheinlich durch die Testsituation als solche weniger gestört sein, sodass sie sich besser auf die eigentliche Aufgabe fokussieren können.

Wenn Sie nicht gerade wissen wollen, wie ganz bestimmte Personengruppen ‚abschneiden', bietet es sich an, einen Mix verschiedener Personen zu den Beobachtungen einzuladen. Hier stellt sich übrigens auch die Frage, wie viele Proband*innen Sie überhaupt brauchen (vgl. Kasten „Wie viele Nutzer*innen brauche ich?")

Wie viele Nutzer*innen brauche ich?

Das Verfahren in diesem Buch ist eine Form qualitativer Forschung. Anders als bei quantitativer Forschung stehen in der Regel keine statistisch signifikanten Ergebnisse im Vordergrund, sondern es geht um Variationen der Frage, wie *bestimmte* Personen mit Ihrer Software umgehen. Dazu können Sie sehr tief gehende Details ermitteln und zahlreiche Erkenntnisse ziehen.

Während eine quantitative Studie erst ab einer Zahl von wenigstens n = 30 Teilnehmer*innen langsam aussagekräftig wird, kommen Sie bei qualitativen Studien mit weniger Personen aus. Oft reichen schon fünf bis sechs Proband*innen, um die wesentlichen Probleme oder offenen Fragen der zu beobachtenden Situation zu identifizieren. Planen Sie aber einen ‚Puffer' ein, d. h. planen Sie lieber mit sieben bis acht Personen, falls einige von ihnen ausfallen oder ihre Einwilligung zur Untersuchung später zurückziehen (vgl. Abschn. 4.4.2).

Trotz der geringen Teilnehmer*innenzahl ist der Auswertungsaufwand bei qualitativen Studien höher. Einen breit gestreuten Fragebogen können Sie in kurzer Zeit auswerten, aber bei qualitativen Studien müssen Sie sich selbst durch das Material arbeiten und es interpretieren. Eine Studie mit acht Personen, mit Beobachtungssitzungen zu ca. 20 Minuten, kann in der Auswertung durchaus zwei Wochen Arbeit für eine Person bedeuten, je nach Fragestellung.

Nachdem Art und Anzahl der Proband*innen geklärt ist, können Sie sich auf die Suche nach geeigneten Proband*innen machen. Hier sind einige Möglichkeiten, die Sie natürlich wieder nach Bedarf variieren können:

- *Rekrutieren Sie Ihre Kolleg*innen.* Dies eignet sich, wenn Sie während der Arbeit an einem Programm oder einer neuen Programmversion ein Feedback brauchen. Lassen Sie Ihre Kolleg*innen zum Beispiel eine ausgewählte Funktion oder ein neues Element der Benutzeroberfläche für zehn Minuten ausprobieren und kommentieren, während Sie einen Screenrekorder laufen lassen und ein Mikrofon (notfalls geht ein Smartphone!) auf den Schreibtisch legen.
- *Suchen Sie studentische Proband*innen aus einem Fachbereich, der Ihnen passend erscheint.* Für größere, weniger spontane Beobachtungen können Sie per Aushang oder

Nachricht in sozialen Medien Studierende von Universitäten und Hochschulen gewinnen – besonders, wenn Sie bereit sind, eine Aufwandsentschädigung zu zahlen. Je nach Projekt können die so gewonnenen Proband*innen das vielleicht sogar von zu Hause aus machen (vgl. dazu aber die Hinweise im Kasten „Den Beobachtungsort festlegen"). Da es sich aber um Personen handelt, die mit Ihrem Projekt vermutlich vorher nichts zu tun hatten und vielleicht nicht den Anwendungsbereich kennen, wird es nötig sein, eine adäquate Einführung zu geben und klare Aufgabenstellungen zu formulieren.

- *Fragen Sie Ihre Nutzer*innen.* Heute wird gerne ein Feedback-Link in Software eingebaut; Nutzer*innen können dann ihre spontanen Eindrücke und Fehlerberichte verschicken, die dann hoffentlich die Entwickler*innen erreichen und von ihnen bearbeitet werden. Einen Schritt weiter gehen Sie, wenn Sie die Nutzer*innen Ihres Produkts für die Beobachtungen gewinnen. Der Vorteil ist, dass diese Personen aus dem Anwendungsbereich kommen, für den Sie Ihre Software entwickeln und sie Ihnen so sehr praxisnahe Eindrücke vermitteln können.

4.3.2 Den Beobachtungsort festlegen

Die Auswahl des Beobachtungsortes kann nicht ganz von der Fragestellung, den benötigten Daten und den Proband*innen getrennt werden. Es hängt aber auch von Ihren Möglichkeiten ab.

- *Die Proband*innen kommen zu Ihnen.* Wenn Sie in einer Großstadt tätig sind, genug ungestörten Platz zur Verfügung und zeitlich flexible Proband*innen gewonnen haben (zum Beispiel Student*innen), können Sie die Beobachtungssitzungen vielleicht bei sich vor Ort durchführen. Unter Umständen darf Ihre Software auch noch gar nicht Ihr Haus verlassen – dann müssen die Proband*innen zu Ihnen kommen. Der Vorteil ist, dass Sie ideale Beobachtungssituationen schaffen können.
- *Sie kommen zu den Proband*innen.* Wenn es nicht möglich ist, dass Sie die Beobachtungssitzungen bei sich durchführen (vielleicht liegt Ihr Unternehmen abgelegen im ländlichen Raum oder die normalen Betriebsabläufe würden zu sehr gestört), können vielleicht Sie zu den Proband*innen kommen – wenn Sie beispielsweise Software für ein anderes Unternehmen entwickeln, könnten Sie dort, sozusagen in-house, mit Testnutzer*innen zusammenkommen. Ideal wäre es, wenn Sie sogar direkt am künftigen Einsatzort des Produkts beobachten könnten, sofern sich das logistisch und ethisch (vgl. Abschn. 4.4) vertreten lässt, aber ein spezieller Raum ist ebenso möglich.
- *Sie nutzen das Internet.* Wenn es nicht anders geht, können Sie auch das Internet zu Hilfe nehmen. Sie weisen Ihre Proband*innen schriftlich, telefonisch oder via *Skype* in die zu erledigenden Aufgaben und in die zu nutzende Software ein und lassen Ihre Proband*innen die Aufgaben von deren Arbeitsplatz oder von zu Hause aus erledigen. Allerdings will das gut überlegt sein. Der erste Gedanke ist oft, dass die Proband*innen

sich die zu untersuchende Software und die zum Aufzeichnen der Beobachtungen nötigen Programme installieren, selbst bedienen und am Ende die Aufzeichnungen an Sie übermitteln. Allerdings besteht das Risiko, dass Ihre Proband*innen abgelenkt sind, dass Hard- und Software der Proband*innen nicht geeignet ist, dass sie Fehler beim Bedienen der Aufzeichnungssoftware machen oder sogar aufgezeichnete Daten manipulieren – das heißt, Sie haben wenig Kontrolle über die Situation. Sinnvoller ist es daher, wenn Ihre Proband*innen sich bei Ihnen auf einen speziell vorbereiteten Testrechner aufschalten (dafür reicht schon ein einfaches Tool wie *TeamViewer* oder ähnliches). Die Proband*innen bedienen die Software also auf Ihrem Testrechner und Sie selbst schneiden die Audiodaten und die Bildschirminhalte mit.

4.3.3 Benötigte Technik

Die für Beobachtungen nötige Technik hängt von den benötigten Daten und dem gewählten Beobachtungsort ab:

- *Audiorecorder für Audiodaten:* Die Aufzeichnung der Kommentare von Nutzer*innen während der Nutzung (das schon erwähnte ‚laute Denken') ist das wichtigste Element des hier beschriebenen Verfahrens. Sie benötigen also eine Möglichkeit, diese Kommentare in guter Qualität aufzuzeichnen, aber so, dass sich Ihre Proband*innen von dem Aufzeichnungsgerät nicht gestört fühlen – gegebenenfalls ist daher ein Smartphone, das beiläufig auf dem Tisch liegt, geeigneter als ein Diktiergerät oder hochwertige Mikrofontechnik.
- *Kameras für Videodaten:* Wenn Sie auch die Handlungen und die Gesichtsausdrücke Ihrer Proband*innen aufzeichnen wollen, ist eine Kamera nötig. Auch hier ist eine dezent angebrachte USB-Kamera oder in den Bildschirm eingebaute Webcam mitunter besser geeignet als teure, aber zu dominante Videotechnik.
- *Screenrekorder für Bildschirminhalte:* Wenn Sie die Bildschirminhalte selbst mitschneiden wollen, eignet sich Screenrecorder-Software. Sie sollte so konfiguriert sein, dass Bedienvorgänge flüssig aufgezeichnet werden, ohne Ruckler und Auslassungen (wenn beispielsweise ein*e Proband*in hektisch-unentschieden die Maus zwischen zwei Bereichen der Benutzeroberfläche hin und her bewegt, könnte das wichtig sein, aber verloren gehen, wenn bei der Aufzeichnung Frames ausgelassen werden).
- *Logfiles für Softwareaktivität:* Manchmal kann es auch nützlich sein, die Softwareaktivität genau zu protokollieren. Dazu können Sie eine spezielle Testversion der Software erstellen, die etwa bei jedem Funktions- oder Methodenaufruf, zu Beginn und Ende von zeitintensiven Schleifen und anderen kritischen Momenten einen aussagenkräftigen Kommentar mit Zeitstempel (Minuten und Sekunden) in eine Logdatei schreibt. Das können Sie dann später mit den anderen Daten, die Nutzer*innenaktivität zeigen, abgleichen. Wie fein- oder grobgranular Sie das machen wollen, hängt von Ihrer Fragestellung ab.

4.3.4 Die richtigen Aufgaben stellen

Sie haben Ihre Fragestellung definiert und geeignete Proband*innen ausgewählt. Damit Sie durch die Beobachtung dieser Proband*innen gut nutzbare Ergebnisse erhalten, müssen Sie den Proband*innen passende Aufgaben stellen. Die Proband*innen versuchen dann, diese Aufgaben unter Nutzung der jeweiligen Software umzusetzen und Sie beobachten sie dabei mit Hilfe geeigneter technischer Aufzeichnungsverfahren (vgl. Abschn. 4.3.3).

Entscheidend ist, dass die Aufgabenstellung von den Proband*innen verstanden wird. Sie darf in sich keine Fragen oder Zweifel aufkommen lassen, denn dann lenkt die Irritation durch die Aufgabe Ihre Proband*innen ab. Wenn die Aufgabe falsch oder nicht vollständig verstanden wird, beeinflusst das die Nutzung der Software. Dann erhalten Sie unter Umständen Daten, die nicht für Ihre Fragestellung geeignet sind. In den Begriffen klassischer Gütekriterien: Die erhobenen Daten wären nicht valide, weil die Beobachtung nicht das ‚misst', was Sie ‚messen' wollten.

Eine gute Aufgabenstellung zu entwickeln, ist nicht trivial und vor allem in der Pädagogik gibt es ein großes Forschungsgebiet allein zur Aufgabenentwicklung. Das muss hier nicht ausführlich behandelt werden. Bitte beachten Sie aber folgende Hinweise, die sich ganz grundsätzlich als brauchbar für Entwicklung und Präsentation von Aufgabenstellungen herausgestellt haben:

- Formulieren Sie Aufgaben schriftlich und halten Sie sie auf einem Aufgabenblatt fest, das Sie an Ihre Proband*innen ausgeben.
- Planen Sie genug Zeit zu Beginn jeder Beobachtungssitzung ein, damit die Proband*innen die Aufgaben lesen können. Stehen Sie während dieser Zeit für Nachfragen zur Verfügung.
- Unterteilen Sie Aufgabenkomplexe in Teilaufgaben, die nacheinander bearbeitet werden können.
- Trennen Sie Erklärungen von Anweisungen und Fragen. Wenn Sie etwas erklären müssen, tun Sie dies in einem eigenen Absatz. Fügen Sie keine Erklärungen in die eigentliche Frage oder Anweisung ein.
- Achten Sie auf eine angemessene Reihenfolge der Einzelaufgaben: Beginnen Sie mit einer einfachen Aufgabe, um Ihren womöglich nervösen Proband*innen ein erstes Erfolgserlebnis zu verschaffen; sie werden sich dadurch etwas entspannen. Stellen Sie schwierigere Aufgaben in der Mitte. Da die Konzentration nach einer Weile nachlässt, stellen Sie am Ende des Komplexes wieder einfachere Aufgaben.
- Formulieren Sie einfache Sätze. Bevorzugen Sie Hauptsätze und vermeiden Sie verschachtelte Nebensätze.
- Verwenden Sie Begriffe, die Ihren Proband*innen aus dem eigenen Alltag bekannt und geläufig sind. Fachbegriffe aus dem Anwendungsbereich der Software können angemessen sein, wenn Ihre Proband*innen damit im Alltag zu tun haben. Vermeiden Sie technische Fachbegriffe aus der Softwareentwicklung oder Begriffe, die Sie vielleicht in Ihrem Team nutzen, die aber von Außenstehenden nicht verstanden werden.

Hinsichtlich des letzten Punktes, der Begrifflichkeiten, sei auch kurz das Konzept der sogenannten Operatoren angesprochen. Operatoren sind Verben, z. B. „*nennen* Sie“, „*klicken* Sie“ oder „*öffnen* Sie“. Mit Operatoren können Sie Aufgaben klarer formulieren, weil beim Lesen des Operators gleich klar wird, welche konkrete Handlung von den Proband*innen erwartet wird. Doch damit dieser Vorteil zum Tragen kommen kann, müssen die Operatoren dem Handlungsfeld angemessen sein und den Proband*innen bekannt sein (ansonsten müssen sie vor der Anwendung erst eingeführt und erklärt werden).

Beobachten Sie beispielsweise Nutzer*innen im Umgang mit einer Textverarbeitung, dann werden manche Operatoren anders sein als beim Umgang mit einer Flugsimulation:

- Für eine Textverarbeitung typische Operatoren sind beispielsweise: „*Erzeugen* Sie eine Tabelle.“ oder „*Formatieren* Sie die Überschrift kursiv.“
- Für eine Flugsimulation typische Operatoren sind zum Beispiel: „*Aktivieren* Sie den Transponder.“ oder „*Funken* Sie den nächstgelegenen Flughafen *an*.“
- Für beide Anwendungsbereiche geeignete Operatoren sind etwa: „*Klicken* Sie im Menü auf Datei.“ oder „*Scrollen* Sie ans Ende der Auswahlliste.“

Der letzte Punkt, die übergreifenden Operatoren, sind für beide Anwendungsbereiche geeignet, weil sie Handlungen der Computernutzung selbst ausdrücken, unabhängig von einer konkreten Anwendung – egal ob Textverarbeitung, Flugsimulation, Webbrowser oder Computerspiel, sind *klicken* und *scrollen* zumindest unter heute üblichen Bedienparadigmen überall verbreitet. Die ersten beiden Beispiele waren hingegen spezifisch für die von der jeweiligen Software unterstützten Anwendungsbereiche.

Operatoren werden oft bei vergleichenden Leistungsbewertungen im Bildungsbereich genutzt. Wenn Sie eine größere Zahl unterschiedlicher Nutzer*innen beobachten wollen, können Operatoren ein Weg sein, die Vergleichbarkeit zu erleichtern. Es kann aber aufwendig sein, die für ein Handlungsfeld geeigneten Operatoren zu identifizieren und sicherzustellen, dass Ihre Proband*innen ein einheitliches Verständnis von den Operatoren und den damit erwarteten Handlungen haben. Sie sollten daher je nach Ihrer Fragestellung und dem Umfang Ihrer Untersuchung abwägen, ob Operatoren wirklich nötig sind.

4.4 Beobachtungen durchführen, Daten erheben

4.4.1 Grundsätzliches am Tag der Beobachtung

Nach Hypothesen- oder Konzeptbildung, Organisation der Beobachtungssituation und Formulierung der Aufgaben geht es daran, die Beobachtung durchzuführen – also die Daten zu erheben, die Sie hinterher bezüglich Ihrer Fragestellung auswerten wollen. Aus meiner Erfahrung sollten Sie wenigstens die folgenden drei Bereiche beachten:

- Wenn Sie nicht gerade beobachten wollen, wie Ihre Proband*innen unter Stress mit Ihrer Software zurechtkommen, dann *sorgen Sie für eine möglichst ablenkungsfreie Situation*. Wenn Sie selbst einen ruhigen Ort zur Verfügung stellen können, ist das einfacher, als wenn Sie beispielsweise am echten Arbeitsplatz künftiger Nutzer*innen beobachten wollen. In letzterem Fall sollten Sie die anderen anwesenden Personen informieren, damit sie Störungen vermeiden.
- *Sorgen Sie dafür, dass die zur Beobachtung eingesetzte Technik problemlos funktioniert.* Beispielsweise: Sind alle Akkus aufgeladen (und Ladegeräte oder Netzteile vor Ort)? Sind benötigte drahtlose Geräte stabil verbunden? Ist genug Speicherplatz für alle Aufzeichnungen vorhanden? Funktioniert der Screenrekorder und ist korrekt konfiguriert? Sind Mikrofonaufzeichnungen gut verständlich? Stehen Netzwerk- und Internetverbindungen?
- *Kümmern Sie sich um Ihre Proband*innen.* Wenn Sie eine Uhrzeit vereinbart haben, seien Sie rechtzeitig vor Ort. Erklären Sie die Aufgaben genau und beantworten Sie Verständnisfragen. Strahlen Sie Ruhe und Vertrauen aus, insbesondere, wenn Ihre Proband*innen noch nie an einer Studie teilgenommen haben und daher vielleicht aufgeregt sind. Stellen Sie Wasser, Tee und Kaffee bereit. Sorgen Sie für eine angenehme Raumtemperatur und Luftqualität.

Nun ist es soweit. Ihre Proband*innen sind vor Ort und eingewiesen, alle Fragen zu den Aufgaben sind geklärt, die Aufzeichnungstechnik funktioniert und läuft, und Sie haben den Raum verlassen. Was tun Sie nun?

Sie warten ab. Erliegen Sie bitte nicht der Versuchung, während der Beobachtungssituation andere Arbeiten zu erledigen, zu telefonieren oder in Meetings zu gehen. Seien Sie stattdessen für Ihre Proband*innen als Ansprechpartner im Hintergrund verfügbar (entweder im Nebenraum oder per Telefon). Informieren Sie die Proband*innen vor Beginn der Beobachtungen, wie sie Sie bei Problemen oder bei frühzeitigem Abschluss der Aufgaben erreichen. Klären Sie in dem Zusammenhang auch, wie lange die Proband*innen bei Nutzungsproblemen allein (ohne Ihre Hilfe) versuchen sollten, eine Lösung zu finden, bevor sie sich an Sie wenden (es geht hier darum, ein Gleichgewicht zu finden zwischen Ihrem Forschungsinteresse wie „Was tun die Proband*innen bei Problemen?“ und einer vertrauensvollen Untersuchungsatmosphäre, bei der sich Proband*innen nicht alleingelassen fühlen).

4.4.2 Ethische Überlegungen

Stellen Sie sicher, dass Sie auf ethisch einwandfreie Weise arbeiten. Der wesentliche Aspekt ist hier die zumindest prinzipiell vorhandene Frage, ob Sie Beobachtungen verdeckt oder offen vornehmen. Bei verdeckten Beobachtungen besteht die Hoffnung, dass sich die Proband*innen (die in dem Fall entweder nicht wissen, wofür sie Proband*innen sind, wann sie es konkret sind, oder dass sie es überhaupt sind) natürlicher verhalten als wenn

sie sich der Untersuchungssituation bewusst sind. Gerade bei Beobachtung mittels der Methode ‚Lautes Denken' hat sich aber gezeigt, dass sich Menschen auch mit Einwilligung und Wissen über die Beobachtung nach kurzer Zeit ‚natürlich' verhalten (van Someron et al. 1994, S. 26), also auf ganz ‚normale' Weise die Nutzungsvorgänge kommentieren (ebd., vgl. Donick 2016). Daher gibt es kaum einen Grund, eine Einwilligung für die Beobachtung erst hinterher einzuholen.

Das Einholen der Einwilligung muss transparent und offen sein. Das Erschleichen von Einwilligungen unter Vorspiegelung falscher Tatsachen oder durch psychologische Manipulation von Personen, sowie das Erzwingen von Einwilligungen (etwa in einem Unternehmen unter Drohung von Abmahnung oder Kündigung) ist ethisch nicht in Ordnung und abzulehnen. Brinker und Sager (2006) nennen die folgenden Beispiele für erschlichene und erzwungene Einwilligungen:

- *Partnerorientierter Druck* (ebd., S. 30): Sie nutzen Ihren Status aus, zum Beispiel als Vertreter eines großen Softwareunternehmens, einer anerkannten Forschungseinrichtung, einer großen Organisation u. ä., oder Sie holen sich die Unterstützung ranghoher Personen im Umfeld der gewünschten Proband*innen (Vorgesetzte, Vorbilder usw.), um diese zur Teilnahme an der Untersuchung zu überreden oder zu verpflichten.
- *Sachzwänge* (ebd.): Sie geben den gewünschten Proband*innen das Gefühl, dass sie aus gesellschaftlichen, wissenschaftlichen oder politischen Notwendigkeiten vernünftigerweise nicht gegen die Teilnahme an der Untersuchung sein können. Sie suggerieren ihnen zum Beispiel, dass Ihre Fragestellung eine so große Bedeutung für sicherere Software, besseren Datenschutz, Arbeitsplatzsicherheit trotz weiterer Automatisierung usw. hat, dass sich die gewünschten Proband*innen quasi schuldig machen würden, wenn sie sich einer Teilnahme verweigern.
- *Suggerierte Selbstzwänge* (ebd.): Sie appellieren bei Ihren gewünschten Proband*innen an ein positives Selbstbild, das aber vor allem Ihren Bedürfnissen als Forschende*r entspricht. Sie erzeugen zum Beispiel bei einer Person den Eindruck, dass Sie nie in der Lage sein werden, die Untersuchung zu beenden, wenn die Person sich weigert, ihr Können oder ihr Wissen weiterzugeben. Der suggerierte Selbstzwang besteht hier darin, dass Sie die Person implizit als nicht hilfsbereit, selbstsüchtig usw. darstellen; das setzt die Person unter Druck, durch die Teilnahme an der Untersuchung zu zeigen, dass sie doch hilfsbereit und nicht selbstsüchtig ist.

Sie können auf diese Weisen zwar eine Einwilligung zur Teilnahme an der Untersuchung erlangen, sodass Sie juristisch womöglich nicht so einfach belangt werden können (ebd., S. 31). Aber in ethischer Hinsicht sind solche Manipulationen abzulehnen. Ein Bekanntwerden dieses Vorgehens würde sich zudem negativ auf die Reputation Ihres Projekts oder Unternehmens auswirken: Denken Sie etwa an die Aufregung, die entstand, als im Jahr 2019 bekannt wurde, dass Mitarbeiter*innen von Apple, Google und Amazon die Aufzeichnungen der jeweiligen Sprachassistenten (Siri, Google Assistant, Alexa) auswerten.

Wenn so etwas wiederholt geschieht, stellt dies auch die Legitimation kommunikationswissenschaftlicher Forschung insgesamt in Frage (ebd., S. 25). Daher, aber vor allem im Interesse Ihrer Proband*innen und Ihrer eigenen Person beachten Sie bitte folgende Hinweise, die Sie *vor* der Zustimmung der Proband*innen zur Beobachtung klären müssen:

- Erklären Sie Ihren Proband*innen, in welchem Umfang, in welcher Dauer, an welchem Ort und in welcher Art und Weise Sie die Beobachtungen vornehmen werden. Stellen Sie im Zweifelsfall auch klar, was und wo Sie *nicht* beobachten werden.
- Machen Sie deutlich, in welchen Formaten Sie Aufzeichnungen anfertigen werden, welche Technik Sie dafür nutzen, wie Sie die Aufzeichnungen später auswerten und wer daran beteiligt sein wird. Räumen Sie den Proband*innen die Möglichkeit ein, einzelne Formate oder Techniken abzulehnen (wenn sie sich zum Beispiel nicht filmen lassen wollen, aber zu Audioaufzeichnungen bereit sind).
- Stellen Sie klar, wofür Sie die Aufzeichnungen benötigen bzw. die Auswertungen anfertigen: für die Verbesserung der Software, für die Identifikation von Fehlern, für wissenschaftliche Beiträge bei Tagungen und Zeitschriften, für eine Abschlussarbeit usw. Bedenken Sie auch, dass Sie das Material zukünftig möglicherweise noch für Zwecke nutzen wollen, die Ihnen jetzt selbst noch nicht bewusst sind.

Holen Sie sich die schriftliche Einwilligung Ihrer Proband*innen für die genannten Punkte. Am sinnvollsten ist es, den Proband*innen vor der Zustimmung zur Teilnahme an der Untersuchung die genannten Aspekte in verständlicher Sprache schriftlich auszuhändigen und genug Zeit zum Lesen zu geben. Stehen Sie bereit, um Fragen zu beantworten und mögliche Missverständnisse zu klären. Erst wenn alles geklärt ist, lassen Sie die Proband*innen unterschreiben. Weisen Sie die Proband*innen dabei deutlich darauf hin, dass sie ihre Einwilligung auch jederzeit zurückziehen dürfen. Seien Sie sich bewusst, dass Sie im Fall einer zurückgezogenen Einwilligung Alternativen finden müssen – planen Sie das schon zu Beginn ein.

4.4.3 Was Menschen tun, wenn sie sich beobachtet fühlen

Abschließend möchte ich noch einige Aspekte ansprechen, mit denen Sie trotz bester Vorbereitung der Proband*innen rechnen müssen. Wenn Menschen sich beobachtet fühlen, können sie sich ungewöhnlich verhalten. Man will zum Beispiel etwas besonders gut machen, um nicht als überfordert dazustehen, und macht gerade dann Fehler. Oder man will Ihnen helfen, Ergebnisse zu erzielen, von denen man denkt, Sie würden sie sich wünschen und verdeckt dabei das, um was es Ihnen eigentlich geht. Beides gilt es zu vermeiden.

- *Andere Leistungsfähigkeit unter Beobachtung:* Aus der Forschung zu Leistungsbewertung im Bildungs- und Arbeitskontext weiß man, dass viele Menschen unter Leistungsdruck schlechtere Leistungen erbringen als normalerweise. Selbst einfache Aufgaben können dann als größeres Hindernis wahrgenommen werden. Eine Beobachtungssitua-

tion kann wie Leistungsdruck wirken, denn man möchte nicht als inkompetent, unwissend usw. erscheinen. Die in diesem Kapitel genannten Maßnahmen zur Aufgabenentwicklung und zur Betreuung von Proband*innen dienen dazu, diesen Druck zu verringern – den Proband*innen das Gefühl zu geben, dass sie willkommen sind, dass sie sich keine Sorgen machen müssen. Versuchen Sie, Ihren Proband*innen zu vermitteln, dass Sie nicht deren Leistung als Personen bewerten, sondern dass es Ihnen um die Software geht.

- *Soziale Erwünschtheit:* Oft neigen Menschen in Untersuchungen dazu, so zu antworten oder so zu handeln, wie sie annehmen, dass es gewollt ist. Das kann in unterschiedlicher Form auftreten. Beispielsweise könnten Proband*innen denken, dass Sie als Untersuchungsleiter*in negative Konsequenzen erleiden werden, wenn die getestete Software Fehler aufweist. Um Ihnen ‚zu helfen' (und dadurch vielleicht auch Ihre Sympathie zu erlangen), könnten die Proband*innen dann versuchen, Softwarefehlern aus dem Weg zu gehen oder sie als Bedienfehler zu verschleiern – dabei wären es gerade die Softwarefehler, an denen Sie interessiert sind! Es gibt auch Fälle sozialer Erwünschtheit, die sich auf das soziale Umfeld der Proband*innen beziehen. Bei Beobachtungen am Arbeitsplatz etwa könnten die Proband*innen vorher durch eine*n Abteilungsleiter*in die implizite Anweisung erhalten haben, nicht so viel Zeit zu ‚vergeuden'. Aufgrund der größeren sozialen Nähe verhalten sich diese Proband*innen dann so, wie sie glauben, dass es ihr*e Abteilungsleiter*in wünscht – sie versuchen, die Aufgabenstellungen auf einem anderen (schnelleren) als den von Ihnen beabsichtigen Weg zu lösen, oder sie brechen schon bei geringen Schwierigkeiten ab.
- *Eigene Interessen:* Das vorige Beispiel kann noch eine andere Komponente enthalten. Vielleicht soll in der Abteilung einer Firma eine neue Softwareversion eingesetzt werden, mit der sich Arbeitsabläufe verändern würden, und die Mitarbeiter*innen der Abteilung möchten das nicht. Sie könnten die Teilnahme an der Untersuchung dann dafür nutzen, dies deutlich zu machen – nach dem Motto: ‚Wenn festgestellt wird, dass die neue Software schlecht ist, wird sie nicht eingeführt'. Diese Proband*innen könnten dann eigentlich kleinere Probleme zu Grundsatzfragen hochspielen.

Das Problem der Leistungsfähigkeit ist vor allem zu Beginn und am Ende einer Beobachtungssitzung bedeutsam. In der Regel gewöhnen sich Proband*innen nach einigen Minuten an die besonderen Umstände der Situation, entspannen sich und verhalten sich anschließend ‚natürlicher'. Deswegen sollten die einzelnen Sitzungen nicht zu kurz und nicht zu lang sein; die Aufgabenstellungen sollten einfach beginnen, dann komplexer werden und zum Schluss wieder einfach enden (vgl. Abschn. 4.3.4).

Die Probleme sozialer Erwünschtheit und eigener Interessen können nur durch Aufklärung und gute Einweisungen bearbeitet werden. Vermitteln Sie Ihren Proband*innen, worum es Ihnen geht und warum das wichtig ist. Machen Sie deutlich, warum es Ihnen mehr hilft, wenn die Proband*innen die Software und die Nutzungssituation so nehmen, wie sie eben kommen, anstatt sie in falsch verstandener Hilfsbereitschaft oder aufgrund eigener Interessen zu manipulieren.

Literatur

Alheit, Peter: „Grounded Theory“: Ein alternativer methodologischer Rahmen für qualitative Forschungsprozesse. Manuskript, Universität Göttingen, Göttingen 1999.

Ammer, Daniela: Die Umwelt des World Wide Web. Bildung für nachhaltige Entwicklung im Medium World Wide Web aus pädagogischer und systemtheoretischer Perspektive. Dissertation, Universität Tübingen, Tübingen 2008.

Baecker, Dirk: Form und Formen der Kommunikation. Frankfurt/Main 2007.

Brinker, Klaus/Sager, Sven F.: Linguistische Gesprächsanalyse. Eine Einführung. Berlin 2006.

Bergmann, Jörg R.: Ethnomethodologische Konversationsanalyse. In: Gerd Fritz/Franz Hundsnurscher (Hrsg.), Handbuch der Dialoganalyse. Tübingen 1994, S. 3–16.

Donick, Mario / Tavangarian, Djamshid: Qualitatives Evaluationsverfahren für interkulturelle E-Learning-Szenarien am Beispiel des Online-M.Sc „Visual Computing“. In: Apostolopoulus, Nicolas / Mußmann, Ulrike / Rebensburg, Klaus / Schwill, Andreas / Wulschke, Franziska (Hrsg.): Grundfragen Multimedialen Lehrens und Lernens. E-Kooperationen und E-Praxis. Tagungsband GML² 2010, 11.-12. März 2010, Berlin 2010, S. 273–285.

Donick, Mario / Daher, Robil / Schwelgengräber, Wiebke / Krohn, Martin / Tavangarian, Djamshid: Messung und Optimierung der Interaktionsgüte (Quality of Interaction) zeitverzögerter CSCW-Anwendungen. In: Praxis der Informationsverarbeitung und Kommunikation, Band 35, Heft 2, 2012, S. 107–112.

Donick, Mario: „Offensichtlich weigert sich Facebook, mir darauf eine Antwort zu geben“: Strukturelle Analysen und sinnfunktionale Interpretationen zu Unsicherheit und Ordnung der Computernutzung. Hamburg 2016.

Engemeier, Norbert / Hauswald, Rico / Schubbe, Daniel: Wissenschaftstheorie. In: Breitenstein, Peggy / Rohbeck, Johannes: Philosophie. Geschichte – Disziplinen – Kompetenzen. Stuttgart / Weimar 2011, S. 165–180.

Feyerabend, Paul: Wider den Methodenzwang. Frankfurt/Main 1986.

Hill, Richard J./Crittenden, Kathleen Stones: Proceedings of the Purdue Symposium on Ethnomethodology. West Lafayette 1968.

Huth, Radoslaw Miroslaw: Rational Choice und Altruismus. Hilfsbereitschaft am Beispiel der Teilnahme an wissenschaftlichen Interviews. Dissertation, RWTH Aachen, Aachen 2008.

Jürgens, Eiko: Leistung und Beurteilung in der Schule. Eine Einführung in Leistungs- und Bewertungsfragen aus pädagogischer Sicht. Sankt Augustin 2005.

Krapp, Andreas/Weidenmann, Bernd (Hrsg.): Pädagogische Psychologie. Ein Lehrbuch. Basel 2006.

Klauer, Karl Josef: Forschungsmethoden der Pädagogischen Psychologie. In: Krapp, Andreas / Weidenmann, Bernd (Hrsg.): Pädagogische Psychologie. Ein Lehrbuch. Weinheim / Basel 2006, S. 75–98.

Koller, Hans-Christoph: Interpretative und partizipative Forschungsmethoden. In: Faulstich-Wieland, Hannelore / Faulstich, Peter (Hrsg.): Erziehungswissenschaft. Ein Grundkurs. Reinbek bei Hamburg 2008, S. 606–621.

Luhmann, Niklas: Die Wissenschaft der Gesellschaft. Frankfurt/Main 1992.

Mayring, Philipp: Qualitative Inhaltsanalyse. Weinheim / Basel 2002.

Suchman, Lucy: Human-Machine Reconfigurations. Plans and Situated Actions 2nd Edition. Cambridge 2007.

Strübing, Jörg: Grounded Theory. Zur sozialtheoretischen und epistemologischen Fundierung des Verfahrens der empirisch begründeten Theoriebildung. Wiesbaden 2008.

van Someron, Maarten W./Barnard, Yvonne F./Sandberg, Jacobijn A.C.: The Think Aloud Method. A practical guide to modelling cognitive processes. London 1994.

Spencer-Brown, George: Laws of Form. Leipzig 2011.

Sutter, Tilmann: Medienanalyse und Medienkritik. Forschungsfelder einer konstruktivistischen Soziologie der Medien. Wiesbaden 2010.

Strukturelle Analyse der Beobachtungsdaten

5

Inhaltsverzeichnis

5.1 Transkription und Aufbereitung

Der erste Schritt der Datenauswertung besteht darin, das aufgezeichnete Material in eine handhabbare Form zu bringen. Während Logdateien von Software bereits in schriftlicher Form vorliegen, muss Audio- und Videomaterial erst in die Schriftform überführt werden. Dies bezeichnet man als *transkribieren* und das entstehende Dokument als *Transkript*. Das Transkript ist das Dokument, mit dem Sie vorwiegend arbeiten – es ist im ganzen Forschungsprozess fast das wichtigste Element: Sie erstellen es, markieren darin nach einem bestimmten Schema (vgl. Abschn. 5.3.1, 5.3.2 und 5.3.3) Auffälligkeiten, setzen es in Beziehung zu anderen Materialien und interpretieren Transkript und Beziehungen hinsichtlich Ihrer Fragestellung.

Da es heute gut funktionierende Spracherkennungssoftware gibt, liegt die Idee nahe, die Transkription automatisch durchführen zu lassen. Dies wird aber in vielen Fällen nicht funktionieren, denn Ihre Proband*innen sprechen meist nicht so laut und deutlich, wie das nötig wäre. Sie verschlucken Silben, verhaspeln sich, setzen erneut an, lassen Sätze unbeendet, sind zu leise oder nuscheln. Das sind normale Verhaltensweisen beim Sprechen, die wir im Alltag

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8_5

nur wahrnehmen, wenn sie wirklich die Verständigung erschweren. Doch für die Untersuchung von Kommunikation sind sie wichtige Marker. Im Falle ‚lauten Denkens' während der Softwarenutzung können sie zum Beispiel auf Nutzungsprobleme hindeuten. Eine automatische Spracherkennung würde diese Marker bestenfalls einebnen, schlimmstenfalls würde die Erkennung gar nicht zu brauchbaren Ergebnissen führen. Deswegen kommen Sie nicht umhin, die Transkription der Audiodaten durch Menschen vornehmen zu lassen.

Daher kann die Transkription auch einer der aufwendigsten Aspekte des Verfahren sein, je nachdem, wie viele Details Sie bewahren wollen. Es gibt Annotationsschemata, mit denen Sie kleinste Unterschiede in der Stimmführung (Prosodie) oder unscheinbarste nonverbale Signale festhalten können, in anderen wird vor allem der Inhalt transkribiert. Wiederum hängt es von Ihrer Fragestellung ab, für welchen Detailgrad Sie sich entscheiden. Im Kasten „Einfaches Annotationsschema & Beispiel" zeige ich einen Vorschlag, den ich in Donick (2016) verwendet habe, um den Umgang von Nutzer*innen mit Software zu beobachten; Sie können es gern für Ihre Zwecke erweitern oder anpassen. Einen zwar für die gesprächslinguistische Forschung gedachten, aber dennoch nützlichen Überblick über detailliertere Transkriptionsverfahren finden Sie bei Brinker und Sager (2006, S. 42–56).

Einfaches Annotationsschema & Beispiel

In Donick (2016) sowie in den Transkripten des vorliegenden Buches werden Audioaufzeichnungen so verschriftlicht, wie sie gesprochen werden. Das bedeutet, dass offensichtliche sprachliche Besonderheiten bewahrt werden. Verschluckte Endungen, Füllwörter, falsche Wörter, falsche Grammatik und so weiter werden so festgehalten, wie sie eben auftreten. Daneben verwende ich bei der Transkription folgende Annotationen:

.	sehr kurze Pause beim Sprechen (max. 1 Sekunde)
…	kurze Pause beim Sprechen (2 Sekunden)
(… 3s)	längere Pause mit Sekundenangabe
[…]	Auslassung bei Herstellung oder Darstellung des Transkripts
[Wort?]	schwer verständliche Äußerung
GROSS	besonders betonte oder laute Äußerung

Mit diesem Schema ist beispielsweise folgendes Transkript (aus Donick 2016, S. 283) möglich:

1. wenn ich drucken will, dann erscheint … ja … son Zeichen, dass ich das hier drucken müsste
2. Zieht aber auch nicht ein
3. Und denn … Ja anderswo wüsst ich nicht wo ich das
4. Ich wüsste allerdings auch gar nicht, wo soll das da rauskommen, wenn ich das hier reinstell

Zur besseren Referenzierbarkeit werden die Zeilen (Segmente) des Transkripts nummeriert. Wenn mehr als eine Person an der aufgezeichneten Situation beteiligt war, nennen Sie dann den (anonymisierten) Bezeichner der Person. Die Entscheidung, ob für eine Äußerung eine neue Zeile begonnen wird, kann sich zum Beispiel an Satzstruk-

turen oder an semantisch abgrenzbaren Propositionen (Aussagen) orientieren. Hier können Sie auch durchaus Ihrem Sprachgefühl vertrauen.

Der Vorgang der Transkription besteht nun ‚einfach' darin, dass Sie sich die gesprochenen Audiodaten anhören und während des Hörens abtippen. Entsprechend dem gewählten Annotationsschema machen Sie bei Bedarf schon beim Abtippen die nötigen Markierungen. Während des Tippens werden Sie feststellen, dass Sie schon dabei weitere Vorstellungen zu Ihrer Fragestellung entwickeln. Halten Sie diese ruhig fest; sie können als Arbeitshypothese oder als Verfeinerung Ihrer sensibilisierenden Konzepte (vgl. Abschn. 4.2) fungieren.

5.2 Die Globalstruktur ermitteln: Sich einen Überblick verschaffen

In der strukturellen Analyse der nunmehr schriftlich vorliegenden Daten untersuchen Sie die sprachliche, paraverbale und nonverbale Struktur der Transkripte. Dabei unterscheidet man Globalstruktur und Lokalstruktur (Donick 2016, S. 166; vgl. Brinker und Sager 2006, S. 143 ff., 163 ff.). Die Globalstruktur betrifft den gesamten Verlauf der Beobachtung; um diese Struktur herauszuarbeiten, unterteilen Sie das Material in zusammengehörige Abschnitte, etwa hinsichtlich des Themas. Dadurch verschaffen Sie sich bereits einen Überblick über den Verlauf der beobachteten Nutzungssituationen. Die Lokalstruktur (vgl. Abschn. 5.3) betrifft sprachliche und thematische Details innerhalb der einzelnen Abschnitte. Beides schauen wir uns im Folgenden an.

Bei der Analyse der Globalstruktur geht es im Prinzip darum, eine Gliederung für die transkribierte Beobachtungssituation zu ermitteln. Im einfachsten Fall halten sich Ihre Proband*innen klar an die Reihenfolge der Aufgaben, die Sie ihnen gegeben haben (vgl. Abschn. 4.3.4). Manchmal werden sie davon abweichen, oder es lassen sich innerhalb einzelner Aufgabenschritte längere Untereinheiten identifizieren. Ich schlage vor, dass Sie die Globalstruktur deshalb in zwei Durchläufen ermitteln:

1. Meiner Erfahrung nach hilft es, zuerst die Absichten der beobachteten Person als Grundlage zu nehmen. Beim ‚lauten Denken' verbalisieren Proband*innen ihre Absichten, das heißt, sie teilen mit, was sie als nächstes tun möchten. Häufig drücken sie auch sprachlich aus, wenn sie mit etwas fertig geworden sind (das kann auch nur ein ausgerufenes „so!" sein). Unterteilen sie das Transkript also an den Segmenten (Zeilen), in denen Proband*innen äußern, einen neuen Arbeitsschritt zu beginnen, eine Aufgabe beendet zu haben oder bei der Nutzung nicht mehr weiterzukommen.
2. Anschließend schauen Sie sich die entstandenen Abschnitte erneut an und achten darauf, ob sie in Unterabschnitte unterteilt werden können. Wenn Ihre Proband*innen zum Beispiel die Aufgabe hatten, einen Brief zu schreiben und zu drucken, dann könnte diese Aufgabe vielleicht in die Schritte des Schreibens, des Formatierens und des Ausdruckens unterteilt werden – wenn die Person so vorgeht. Wenn die Person hingegen Schreiben und Formatieren vermischt (beides jeweils im Verbund macht), wäre dies keine geeignete Gliederungsgrundlage, da die entstehende Gliederung viel zu kleinteilig wäre.

Tab. 5.1 Ausschnitt aus einer Globalstruktur (aus Donick 2016, S. 179)

Abschnitt	Arbeitsschritt	Länge	Einleitungs-Segment	Schluss-Segment
1	Anlegen Benutzerkonto	92	So ich fange an jetzt mit [Name des Produkts] zu arbeiten	[Unterbrechung] Dann gucken wir mal hier
2	Profil aufrufen und bearbeiten	56	Nun ist die Aufgabe	So
3	…	…	…	…

Wenn Sie die Globalstruktur ermittelt haben, sollten Sie sie in einer Übersicht festhalten, damit Sie sich bei der weiteren Arbeit daran orientieren können (vgl. Tab. 5.1).

5.3 Die Lokalstruktur analysieren: In die Tiefe gehen

Bei der Analyse der Lokalstruktur gehen Sie in die einzelnen Abschnitte und Unterabschnitte, die Sie vorher herausgearbeitet haben. Das heißt, Sie schauen jetzt nicht mehr im Ganzen auf das Transkript, sondern arbeiten es Segment fü Segment, manchmal auch Wort für Wort, durch. Sie setzen Segmente zueinander in Beziehung und ermitteln die Art der Beziehung. Mitunter setzen Sie auch die Segmente eines Transkripts mit den Segmenten eines anderen Transkripts oder einer Logdatei in Beziehung – je nachdem, welche Daten Sie erhoben haben und welche Fragen Sie beantworten wollen. Auf jeden Fall geht es darum, bestimmte sprachlich und kommunikativ wirksame Formen zu identifizieren, die in der beobachteten Nutzungssituation Bedeutung haben könnten. (*Könnten*, weil Sie nicht mit Bestimmtheit sagen können, worin deren Bedeutung – oder: ihr Sinn – für Ihre Proband*innen liegt. Aber Sie können im letzten Schritt (Kap. 6) die Segmente und ihre Relationen als ‚Spuren' lesen, um Verfahren der Sinnkonstruktion ihrer Proband*innen im Zusammenspiel mit der Software zu identifizieren. Und das ist nötig, wenn Sie – unabhängig von der ganz konkreten Forschungsfrage – wissen wollen, ob und welche Leistungen Ihre Software in der Nutzungssituation erbringt, vgl. Kap. 1 und 3).

Es gibt verschiedene Blickwinkel, unter denen Sie ein Transkript oder seine Beziehungen untersuchen können:

- Die *Themenentwicklung* (thematische Progression) im Transkriptverlauf chronologisch nachvollziehen – was tun Nutzer*innen und Software miteinander?
- Den *Zusammenhang* (Kohärenz) der Relation aller Segmente untereinander ermitteln – wie konsistent ist das, was Nutzer*innen und Software miteinander tun?
- Die *Differenzsetzung* der realisierten Nutzungsschritte als Ausschließen von Alternativen durch die Proband*innen untersuchen – zeigt die Beobachtung das Aufbauen, Bestehen oder Brechen einer strukturellen Kopplung von Nutzer*in und Software im Sinne von System und Umwelt?

Keine der drei Perspektiven ist ‚besser' als die andere. Wie Sie an den Fragen sehen können, eignen sie sich für unterschiedliche Fragestellungen, und sie können sich entsprechend auch ergänzen.

5.3.1 Themenentwicklung (Progression)

Im Alltagsverständnis bezeichnet man mit dem Begriff „Thema" meist den Kern dessen, worum es in einem Äußerungszusammenhang geht. Die Textlinguistik spricht vom Kern des Textinhalts. Eine Äußerung kann in unterschiedlicher medialer Form vorliegen – als mündlicher Redebeitrag, als Lesetext, aber auch als multimediales Konglomerat verschiedener Medien. In diesem Sinn kann auch ein Dialogfeld einer Software ein Thema haben, das ursprünglich durch die Entwickler gesetzt wurde und in der Nutzungssituation durch Rezipient*innen aufgegriffen wird (oder, im Störungsfall, auch nicht). Beispielsweise hat das Fenster zum Speichern eines Dokuments in einer bekannten Textverarbeitungssoftware ein anderes Thema als die Registerkarte „Layout" in derselben Software. Das Thema der Ziel-Eingabeseite eines GPS-Geräts in Auto oder Flugzeug ist ein anderes als das Thema der während der Reise sichtbaren Karte auf demselben GPS-Gerät.

Bevor Sie weiterlesen, denken Sie bitte an Software, die Sie selbst entwickelt haben:

1. Welche Themen lassen sich in Ihrer Software an der Benutzeroberfläche abgrenzen?
2. Anhand welcher konkreten Gestaltungsmittel lässt sich diese Abgrenzung (aus 1.) erkennen?

Wollen wir nun mittels des ‚lauten Denkens' herausfinden, wie Nutzer*innen mit einer Software umgehen, dann sind das Thema und dessen Entwicklung (die thematische Progression) im Rezeptions- bzw. Nutzungsverlauf eine Zugriffsmöglichkeit. Beim ‚Lauten Denken' sprechen Nutzer*innen während der Nutzung über das, was sie sehen, was sie vorhaben und was sie tun; dies haben Sie im Transkript festgehalten. Gleichzeitig liegen für die Nutzer*innen sichtbare Äußerungen auf Softwareseite vor (nämlich in Form der sicht- oder hörbaren Benutzeroberfläche). Mitunter sind auch Logdateien verfügbar, die die normalerweise unsichtbare Aktivität der Software festhalten. In der Analyse lassen sich für alle herangezogenen Daten materiale Strukturen identifizieren (vor allem sprachliche Mittel), anhand derer das Thema bestimmt werden kann. So finden wir heraus, was Nutzer*innen und Software zu einem Zeitpunkt gerade als Thema sehen oder setzen. Dies allein erlaubt schon Aufschluss darüber, ob Software und Nutzer*in einander ‚verstehen' und lässt praktische Rückschlüsse für die weitere Entwicklung zu (allerdings ist es dennoch lohnend, nach der bloßen Ermittlung der Themen auch nach weiteren Zusammenhängen zu suchen, vgl. Abschn. 5.3.2 und 5.3.3).

Um Themen zu ermitteln, können wir uns text- und gesprächslinguistischer Verfahren bedienen, die Themen als Verknüpfung sprachlicher oder semantischer Art herausarbeiten. Die gibt es in unterschiedlichen Varianten, sie laufen aber meist auf zwei wesentliche Punkte hinaus (vgl. Brinker und Sager 2006, S. 78 f.), die im Folgenden erläutert werden.

Verknüpfung mit Wiederaufnahme

Wir identifizieren konkrete Formen als sprachlich zusammengehörig. Wiederaufnahmen liegen vor, wenn einmal eingeführte Wörter wiederholt werden bzw. wenn auf eingeführte Begriffe mittels verschiedener Proformen (vgl. Brinker 2005, S. 33) Bezug genommen wird (Referenzidentität). Wiederaufnahmen liegen ebenfalls vor, wenn eingeführte Begriffe durch Begriffe eines verwandten Kontexts aufgegriffen werden (Kontiguitätsverhältnis), wobei man unter anderem logische (begriffliche Beziehungen), ontologische (naturgesetzliche Beziehungen) und kulturell begründete Kontiguität unterscheidet (ebd., S. 37). Kontiguität liegt vor, wenn sie sich aus dem Sprachsystem selbst begründet (ebd., S. 38), etwa aus von den Sprecher*innen der Sprache akzeptierten und geteilten Bedeutungen von Begriffen.

Der grundlegende Unterschied von Referenzidentität und Kontiguitätsverhältnis sei an den einfachen Beispielen 5.1. und 5.2 erläutert:

Beispiel 5.1

1	Jack:	Ist noch Kaffee da?
2	Jill:	Der ist alle.

Beispiel 5.2

1	Jack:	Ist noch Kaffee da?
2	Jill:	Wir haben keine Bohnen mehr.

Beide Beispiele zeigen eine für Gespräche typische sogenannte Paarsequenz, hier in der Form Frage – Antwort. Jack fragt seine Freundin Jill, ob noch Kaffee da ist. In beiden Fällen verneint Jill das, ohne aber ein deutliches Wort wie ‚nein' zu benutzen. Stattdessen verwendet sie das Mittel der Wiederaufnahme, um sich auf Jacks Frage zu beziehen:

- In Beispiel 5.1 liegt die Wiederaufnahme als Referenzidentität vor; man nennt dies auch explizite Wiederaufnahme. Das Substantiv „Kaffee" (in Jacks Frage) und das Demonstrativpronomen „der" (in Jills Antwort) beziehen sich auf (= referenzieren) denselben konkreten außersprachlichen Gegenstand, nämlich ‚Kaffee als Getränk'.
- In Beispiel 5.2 liegt die Wiederaufnahme als Kontiguitätsverhältnis vor; man nennt dies auch implizite Wiederaufnahme. Jack und Jill sprechen nicht über denselben außersprachlichen Gegenstand. Während Jack sich mit dem Wort „Kaffee" wiederum ‚Kaffee als Getränk' vorstellt, bezieht sich das Substantiv „Bohnen" in Jills Antwort auf den Rohstoff, aus dem das Getränk herstellbar wäre; dies ist ein ontologisches Kontiguitäts-

verhältnis. Da Jack und Jill und wir aufgrund unserer kulturellen Hintergründe um das logische Verhältnis von Bohnen und Getränk wissen, ist zugleich von einer kulturellen Kontiguität zu sprechen.

Eine detailliertere Diskussion von Formen der Wiederaufnahme und ihrer Funktion für die sprachliche Kommunikation finden Sie bei Brinker (2005, S. 27–49) und bei Brinker und Sager (2006, S. 78–80). Für uns ist wichtig, dass Wiederaufnahmen nicht nur in Gesprächen oder innerhalb von Texten eine Rolle spielen, sondern auch beim Umgang von Menschen mit Medien beobachtbar werden, etwa bei der Nutzung von Software, die von ‚lautem Denken' der Nutzer*innen begleitet wird. Hierzu ein weiteres Beispiel (5.3):

Beispiel 5.3

1	Jack:	So, dann klicke ich mal … auf „Speichern" (… 12s)
2	Jack:	Hm, wieso macht … der jetzt nichts mehr? (… 3s)
3	Jack:	Was? „Die Datei konnte nicht gespeichert werden"? (… 4s)
4	Jack:	„Auf Laufwerk C: ist nicht genug" … „SPEICHERPLATZ vorhanden"?
5	Jack:	Aber der Stick hat doch genug …
6	Jack:	Ach, das KANN doch gar nicht SEIN!
7	Jack:	[seufzt] Jill, ist noch Kaffee da?
8	Jill:	Koch dir selber welchen.

Dieses Transkript zeigt ‚lautes Denken' inklusive einiger Annotationen, wie sie schon vorgestellt wurden. Das Beispiel ist fiktiv, zeigt aber typische Äußerungsformen, wie sie oft vorkommen (vgl. Donick 2016). Jack nutzt eine Software und kommentiert diese Nutzung. Dabei liest er auch Ausgaben der Software vor. Jack will offenbar das gerade bearbeitete Dokument speichern (Zeile 1). Nach einer ungewöhnlichen Wartezeit (Zeile 2) erscheint eine Fehlermeldung (Zeile 3–4), nach der auf Laufwerk C: nicht genug Speicherplatz frei wäre. Jack wendet ein, dass „auf dem Stick" sehr wohl „genug" Speicherplatz frei wäre (Zeile 5). Obwohl eine Interpretation der Bedeutung der Äußerungen während der Analyse der Lokalstruktur noch nicht stattfinden sollte, sei hier doch angemerkt, dass wir als Beobachter*innen schon Jacks Irrtum erkennen: Er wollte offenbar auf einem USB-Stick speichern, hat aber die im PC eingebaute Festplatte oder SSD ausgewählt, was wir bei Windows-Systemen an der Laufwerksbezeichnung C: erkennen. Jack übersieht diesen Fehler jedoch und kommentiert den Vorgang in auffälliger Lautstärke (Zeile 6). Anschließend beendet er die Computernutzung für den Moment und tritt stattdessen in soziale Interaktion mit Jill (Zeile 7–8).

1. Bitte markieren Sie in Beispiel 5.3 alle Formen der Wiederaufnahme, die Sie finden können.
2. Geben Sie an, ob es sich bei den einzelnen Wiederaufnahmen jeweils um Referenzidentität oder Kontiguitätsverhältnisse handelt.
3. Ordnen Sie die Wiederaufnahmen in passende Themenbereiche, indem Sie Überschriften dafür finden.

Sie haben beim Bearbeiten der Aufgaben vielleicht bemerkt, dass eine eineindeutige Lösung nicht zu finden ist. Während Referenzidentitäten noch relativ einfach zu erkennen sind (wenn eben mehrere Wörter denselben außersprachlichen Gegenstand bezeichnen), sind Kontiguitäten bereits eine Interpretationsleistung, die von Ihrem eigenen Hintergrundwissen über Situationen abhängt, die der betrachteten Situation ähnlich sind. Diese Unschärfen sind inhärenter Bestandteil menschlicher, natürlichsprachlicher Kommunikation und können auch bei Kommunikationsanalysen nicht ausgeblendet werden. Wichtig ist, dass Sie sich dieser Unschärfen bewusst sind und sie in der Auswertung der Ergebnisse reflektieren. Meinen eigenen Lösungsvorschlag für die Aufgaben finden Sie in Abb. 5.1. Darin habe ich alle zusammengehörigen Wiederaufnahmen aus Beispiel 5.3 hervorgehoben:

Ich habe drei Gruppen referenzidentischer Begriffe identifiziert:

(1) „Speichern“, „nicht gespeichert“ – das außersprachliche gemeinsame Objekt ist die als Code in der Software implementierte Funktion zum Speichern, die erst auf der Benutzeroberfläche als sichtbare Option und dann in der Fehlermeldung referenziert wird;
(2) „genug“, „genug“ – das außersprachliche gemeinsame Objekt ist hier nicht der tatsächlich vorhandene Speicherplatz auf den Laufwerken, sondern ein kognitives Konzept von ‚genügend Speicherplatz‘;
(3) „Kaffee“, „welchen“ – das außersprachliche gemeinsame Objekt ist ‚Kaffee als Getränk‘.

Daneben habe ich vier Gruppen von Kontiguitätsverhältnissen identifiziert, die sich teilweise mit den referenzidentischen Begriffen überschneiden:

(1) „klicke ich“, „macht der nichts“: Jacks Klicken und die von Jack zunächst wahrgenommene fehlende Reaktion der Software auf das Klicken bezeichnen nicht denselben außersprachlichen Gegenstand. Das Klicken auf eine Benutzeroberfläche ist et-

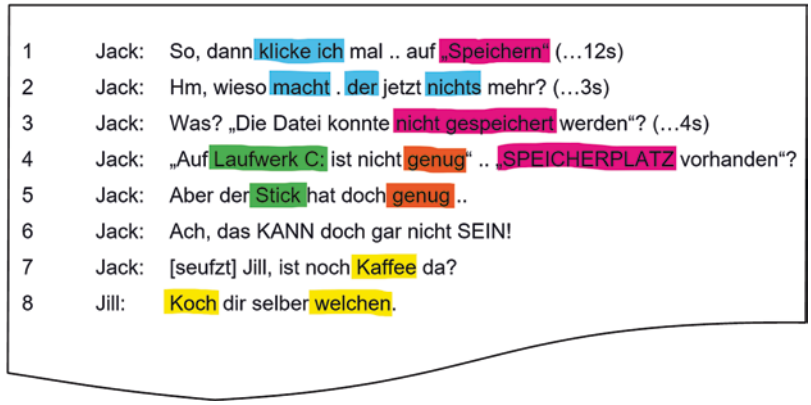

Abb. 5.1 Wiederaufnahmen aus Beispiel 5.3

was anderes als die Funktion, die nach dem Klicken durch die Software ausgeführt wird. Referenzidentität liegt also nicht vor. Dennoch ist offenbar ein Zusammenhang vorhanden. Dieser besteht in einem ontologisch begründeten Kontiguitätsverhältnis, das zum Beispiel in der bekannten Abkürzung „EVA“ (Eingabe-Verarbeitung-Ausgabe) zum Ausdruck kommt und allgemeiner als *Ursache-Wirkung-Erwartung* bezeichnet werden kann. Eine Eingabe wie das Klicken wird mit der Erwartung verbunden, dass nach der Eingabe ‚etwas‘ passiert (idealerweise das, was mit der Eingabe intendiert ist), und dass zu der Aktivität eine Ausgabe als sichtbare Rückmeldung erfolgt. Das ist hier aber nicht der Fall – auf die Eingabe geschieht scheinbar erst mal gar nichts, was Jack entsprechend kommentiert.

(2) „Speichern“, „nicht gespeichert“, „SPEICHERPLATZ“: Die Funktion ‚Speichern‘, der Zustand ‚nicht gespeichert‘ und der ‚Speicherplatz‘ bezeichnen nicht denselben außersprachlichen Gegenstand. Die in der Software implementierte Speicherfunktion ist etwas anderes als der (tatsächlich verfügbare oder kognitiv konzeptualisierte) Speicherplatz, und beides ist etwas anderes als der Zustand ‚nicht gespeichert‘. Aber trotz der fehlenden Referenzidentität besteht ein Kontiguitätszusammenhang. Dieser ist einerseits kulturell begründet (wir wissen, dass man zum Speichern Speicherplatz braucht), andererseits ontologischer Art. Letzterer ist genauer als *tatsächliche Ursache-Wirkung-Beziehung* zu beschreiben: Mit dem Klicken auf „Speichern“ (Ursache) wird die Speicherfunktion ausgelöst (Wirkung). Und das Ausführen der Speicherfunktion (Ursache) führt zu der Fehlermeldung über den ‚nicht gespeicherten‘ Zustand (Wirkung).

(3) „Laufwerk C:“, „Stick“: Beide Begriffe bezeichnen nicht denselben außersprachlichen Gegenstand, es liegt wieder keine Referenzidentität vor. Sie stehen jedoch in einem logischen Kontiguitätsverhältnis, da beide in einer gedachten begrifflichen Oberkategorie ‚Speichermedium‘ enthalten (vgl. Brinker 2005, S. 37), also Hyponyme sind (Brinker unterscheidet Kontiguität von Hyperonymie und Hyponymie, vgl. ebd. 43; ich halte es aber für sinnvoller, diese Kategorien als Varianten logischer Kontiguität zu betrachten).

(4) „Kaffee“, „Koch“, „welchen“: Die vorliegende Referenzidentität von „Kaffee“ und „welchen“ wird ergänzt durch ein ontologisches Kontiguitätsverhältnis, das im Vorgang des ‚Kaffeekochens‘ liegt. Wiederum ist dies auch kulturell begründbar.

Anhand der Referenzidentitäten und der Kontiguitätsverhältnisse lassen sich nun einige Themenbereiche identifizieren, für die ich folgende Überschriften gefunden habe:

(1) speichern nicht möglich (referenzidentische Gruppen 1 und 2; Kontiguitätsgruppen 1, 2 und 3)
(2) falsches Laufwerk (referenzidentische Gruppe 2; Kontiguitätsgruppe 3)
(3) Kaffee kochen (referenzidentische Gruppe 3; Kontiguitätsgruppe 4).

Diese drei Bereiche sind das Thema – der Kern der Äußerungen, die Jack in der Nutzungssituation tätigt, also das, worum es Jack in der Situation geht. Der Kern wurde anhand am Sprachmaterial nachweisbarer Formen (der Wiederaufnahmen) herausgearbeitet.

Anhand der Analyse solcher Formen können Sie in Ihren Beobachtungen bereits Nutzungsschwierigkeiten erkennen und diese nachvollziehbar für andere belegen. Es gibt aber auch Fälle, in denen wir keine Wiederaufnahmen nachweisen können. Doch selbst wenn weder Referenzidentität noch Kontiguität vorliegen, sind wir in der Lage, Zusammenhänge zu erkennen oder zu unterstellen. Oft ist es sogar so, dass bloße Wiederaufnahmen gar nicht ausreichen, um Äußerungen als zusammenhängend zu bezeichnen (ebd., S. 41). Ein weiteres wichtiges sprachliches Mittel sind Konjunktionen und Adverbien (ebd., S. 42), die es in zahlreichen Formen gibt (Konjunktionen wie „und“, „oder“, „aber“, „sowohl – als auch“, „das heißt“, „beziehungsweise“, „denn“, „weil“, „sondern“; Adverbien wie „auch“, „vielmehr“, „also“, „dennoch“, „nämlich“, „später“, „sonst“). Es geht aber auch ganz ohne verknüpfende sprachliche Mittel. Dazu kommen wir jetzt.

Verknüpfung ohne Wiederaufnahme

Schauen wir uns den Dialog an, den Jack und Jill in Beispiel 5.4 führen:

Beispiel 5.4

Jack:	Ist noch Kaffee da?
Jill:	Bist du schon wieder müde?

In dem Beispiel liegt keine Referenzidentität vor – Jill nimmt auf keinen außersprachlichen Gegenstand Bezug, den Jack eingeführt hätte. Ebenso wenig sind am Sprachmaterial logische, ontologische oder kulturelle Kontiguität erkennbar. Es werden auch keine Konjunktionen oder Adverbien zur Verknüpfung verwendet. Dennoch besteht offensichtlich ein Zusammenhang. Mit Hilfe unseres kulturellen Wissens und des situativen Kontexts können Jack und Jill und wir das Wort „müde“ mit der Wirkung des im Kaffee enthaltenen Koffeins in Verbindung bringen, wodurch Jills Gegenfrage verständlich wird.

Brinker (2005, S. 43) bezeichnet diese Fälle als „sprachtranszendent“ (also über das Sprachsystem hinausgehend) und weist darauf hin, dass solche Fälle nur dann verständlich sind, „wenn der Hörer auch über die Kenntnisse verfügt, die der Sprecher bei ihm voraussetzt“ (ebd.). Brinker bezieht sich auf Texte im engeren Sinne, aber sein Hinweis ist für die Entwicklung und Nutzung von Software von größter Bedeutung. Die oft komplexe Gestalt von Software geht über die sichtbare Benutzeroberfläche hinaus, indem die eigentliche Aktivität für die Nutzer*innen unsichtbar geschieht. Ausgaben der Software an der Oberfläche sind nur verständlich, wenn sie sich auf Dinge beziehen, die Nutzer*innen wissen können. Als Entwickler*in ist es Ihre Verantwortung, zu entscheiden, welche Kenntnisse Sie voraussetzen und welche nicht. Schauen wir dazu auf ein letztes Beispiel (5.5). Es ist wieder fiktiv, beruht aber auf zahlreichen im Lauf der letzten zwanzig Jahre so erlebten Situationen.

Beispiel 5.5

1	Jack:	Der … Fortschrittsbalken … „99 Prozent“ (… 6s)
2	Jack:	Da ist der jetzt schon seit fünf Minuten (… 10s)
3	Jack:	Ich brech das jetzt ab … so.

4	Jack:	Hm, „wird abgebrochen“ (… 12s)
5	Jack:	[seufzt] Jill?
6	Jill:	Wir haben nur noch koffeinfreien. (… 5s)
7	Jill:	Bringst du Waschmittel mit?

Jack hat es hier mit einer Software zu tun, die ihren Verarbeitungsfortschritt mit einem Fortschrittsbalken mit Prozentangabe anzeigt. Er steht bei 99 Prozent (Zeile 1), und zwar schon seit „fünf Minuten“ (Zeile 2), was Jack offenbar so sehr irritiert, dass er den ganzen Vorgang abbricht (Zeile 3). Anstatt dies einen sofort sichtbaren Effekt hat, ändert sich zunächst nur der angezeigte Text über dem Fortschrittsbalken (Zeile 4). Irritiert wechselt Jack in den Modus sozialer Interaktion (Zeile 5).

In den Zeilen 1 bis 4 ist das Problem, dass der sichtbare Teil der Software keine ausreichend klare Aussage über ihren tatsächlichen Zustand zulässt. Die Position eines Fortschrittbalkens bei 99 % vermittelt zuerst den Eindruck, dass der betreffende Vorgang gleich abgeschlossen sein müsste. Wenn der Balken nun länger in dieser Position verharrt, passt die Ausgabe offensichtlich nicht mehr zu den Beobachtungen, die Nutzer*innen über den Zustand und die Aktivität der Software machen können. Daraus können unspezifische Fragen entstehen wie: ‚Was braucht der denn jetzt noch so lange?‘, Vermutungen wie: ‚Ist das Programm abgestürzt?‘ oder – mit Luhmann als „Zwang“ (vgl. Luhmann 2009) beschreibbare – Eingriffe wie das Abbrechen des Vorgangs, für das sich Jack hier entscheidet. Ob diese Entscheidung sinnvoll war, oder ob es besser gewesen wäre, zu warten, kann Jack in der Situation offensichtlich nicht beurteilen. Die Software gibt auch keinen Hinweis darauf. Es scheint so zu sein, dass die Entwickler*innen davon ausgingen, dass der Fortschrittsbalken ausreichend Informationen für die Nutzer*innen liefern würde, was aber offenbar nicht der Fall ist. Doch welche weiteren Kenntnisse in der Situation nötig wären, um den tatsächlichen Zustand und die Aktivitäten der Software zu beurteilen (warum es eben bei 99 % nicht weitergeht oder ob es irgendwann noch weitergehen wird), wird aus der Oberfläche nicht deutlich. Dies erzeugt letztlich Verständnisschwierigkeiten und beendet in Jacks Fall die Nutzung.

Jill hat Jack offenbar schon oft genug in solchen Situationen erlebt, dass sie genau weiß, was als nächstes passiert. Obwohl in den Zeilen 5 bis 7 am Sprachmaterial keine Wiederaufnahmen, keine Kontiguitätsverhältnisse, keine Konjunktionen und keine Adverbien für eine Verknüpfung erkennbar sind, sind die Zeilen auf eine über das Sprachsystem hinausgehenden Weise verknüpft. Jill kennt Jack und kann so Erwartungen über seine voraussichtlich nächste Frage bilden. Jill vermutet offenbar, dass Jack sie wieder nach Kaffee fragen wird. Sie wartet seine Frage gar nicht erst ab, sondern gibt ihm direkt den Hinweis, dass es nur noch koffeinfreien Kaffee gibt (Zeile 6). Jills anschließende Bitte, Jack möge Waschmittel mitbringen, drückt implizit Jills Erwartung aus, dass Jack jetzt einkaufen geht (denn Jack will ja Kaffee trinken). Die Verständlichkeit dieser Zeilen hängt stark an dem außersprachlichen Kontext und dem Wissen, dass Jack und Jill übereinander haben (und das wir nur dank der anderen Beispieldialoge nachvollziehen können). Für komplett Außenstehende wären die Zeilen 5 bis 7 vermutlich nur schwer verständlich.

Wozu brauchen wir das?

Aus dem Vorliegen von Verknüpfungen (mit und ohne Wiederaufnahmen) kann ein thematischer Zusammenhang für eine Reihe von Äußerungen erschlossen werden. Die Analyse der Lokalstruktur eines Transkripts besteht nun im Wesentlichen darin, das Material systematisch auf solche Verknüpfungen zu prüfen, diese in sinnvoller Weise auszudrücken und abschließend in übergeordneten Themenbündeln zusammenzufassen. Dadurch identifizieren wir die Themen, die von Nutzer*innen und Software im Zusammenspiel bearbeitet werden, und wir erkennen, ob und wie sich Themen im zeitlichen Verlauf der Nutzung ändern.

Indem Sie anhand der Lokalstruktur von Transkripten ‚lauten Denkens' ermitteln, welche Referenzidentitäten, Kontiguitätsverhältnisse und weiteren Verknüpfungen die Nutzer*innen selbst erkennen oder zuschreiben, erkennen Sie, welche Elemente einer Benutzeroberfläche, welche (echten oder angenommenen) Aktivitäten der Software, sowie welche sonstigen Elemente des situativen Kontexts für die Nutzer*innen relevant erschienen. Und das müssen nicht unbedingt die sein, die Sie als Entwickler*in für die Situation als relevant gesehen hätten. Durch die Analyse der Lokalstruktur erhalten Sie entscheidende Hinweise darauf, ob es Verständigungsprobleme zwischen Nutzer*innen und Software gibt.

Bei der praktischen Arbeit müssen Sie nicht jedes Transkript in der Tiefe analysieren, wie ich es für Beispiel 5.3 dargestellt habe. Dies wird zeitlich kaum möglich sein. Wenn Sie aus derselben Sprachgemeinschaft und einem ähnlichen kulturellen Kontext wie die von Ihnen beobachteten Nutzer*innen stammen, werden Sie viele Beziehungen bereits intuitiv erkennen. Sie wissen nun jedoch, wie Sie Ihre Intuitionen am Sprachmaterial begründen können. Solche Begründungen sind nützlich, wenn Sie gestützt auf Ihre Erkenntnisse zum Beispiel den künftigen Entwicklungsprozess eines Produkts beeinflussen möchten.

Für viele Anwendungszwecke mag dies bereits ausreichen. Es gibt aber noch weitere Möglichkeiten, die Lokalstruktur zu analysieren; die stelle ich Ihnen im Folgenden vor.

5.3.2 Zusammenhang (Kohärenz)

Im vorigen Abschnitt wurde schon der Begriff des „Zusammenhangs" genutzt. Dabei haben wir uns auf den Zusammenhang von aufeinanderfolgenden Äußerungen von Nutzer*innen bezogen und uns angeschaut, wie man solche Zusammenhänge am sprachlichen Material nachvollziehen kann. In diesem Abschnitt erfahren Sie, wie man den Zusammenhang mit Hilfe numerischer Werte ausdrücken kann (statt ihn, wie im letzten Abschnitt, durch Herausarbeiten einzelner Begriffe zu beschreiben). Da diese Methodik als Kohärenzanalyse bekannt ist (vgl. Sucharowski und Schwennigcke 2009; Donick 2016, S. 102–107), verwenden wir von nun an den linguistischen Begriff *Kohärenz* für den Zusammenhang (vgl. Kasten „Kohäsion und Kohärenz").

Kohäsion und Kohärenz

In text- und gesprächslinguistischer Literatur werden Sie mitunter auf die Begriffe Kohäsion und Kohärenz stoßen. Wird diese Unterscheidung vorgenommen, dann bezeichnet man mit Kohäsion die am Sprachmaterial direkt erkennbaren Bindungen, während Kohärenz dann die durch Schlussprozesse erkennbaren Zusammenhänge meint. Kohärenz ist dann die „konzeptuelle Basis" (Linke et al. 2004, 256) hinter oder für die kohäsiven Strukturen. Man spricht dann für die Kohäsion auch von Oberflächenstruktur, für Kohärenz von Tiefenstruktur (ebd.). Ob die Unterscheidung von Kohäsion und Kohärenz nötig ist, ist umstritten; Brinker (2005) hält Kohärenz für entscheidend.

Ablauf der Kohärenzanalyse

Kern der Methodik ist der Vergleich von je zwei Segmenten, um zu bestimmen, ob bzw. zu welchem Grad die beiden Segmente zueinander kohärent (zusammenhängend) sind. Sie setzen also zwei Segmente in Beziehung (Relationierung). Ein Segment kann eine einzelne Äußerung sein, aber auch eine thematisch zusammenhängende Sequenz mehrerer Äußerungen, beispielsweise:

- zwei aufeinanderfolgende Äußerungen Ihrer Nutzer*innen
- eine Nutzer*innen-Äußerung und ein Element der Benutzeroberfläche der Software
- ein Element der Benutzeroberfläche der Software und ein Eintrag einer Logdatei der Software.

Damit die Methodik funktioniert, sollten Sie ein wenig Erfahrung mit dem zu analysierenden Material haben. Außerdem ist es hilfreich, wenn Sie die im vorigen Abschnitt gezeigten Verknüpfungsformen kennen und sicher anwenden können (d. h. explizite Wiederaufnahme als Referenzidentität, implizite Wiederaufnahme als logische, ontologische und kulturelle Kontiguitätsverhältnisse, Konjunktionen, Adverbien sowie über das Sprachsystem hinausgehende, kontextuelle Verknüpfungsformen). Anstatt diese Mittel aber in Transkripten zu markieren, herauszuschreiben oder zu beschreiben, vergeben Sie für eine erkannte Relation zweier Segmente einen numerischen Code.

In Fortführung von Donick und Tavangarian (2010) schlage ich dafür eine pragmatische Codierung vor, die Ihrer Fähigkeit vertraut, die wesentlichen Zusammenhänge am beobachteten Material erkennen zu können, sofern es Ihrer Sprachgemeinschaft und einem ähnlichen kulturellen Kontext entstammt. In diesem Sinne lässt sich die Relation zweier Segmente wie in Tab. 5.2 bewerten.

Die vollständige Bewertung aller möglichen Segmentpaarungen kann zum Beispiel als Matrix dargestellt werden, d. h. als Tabelle, in der die Segmente in die Spaltenköpfe (also die 1. Zeile) fortlaufend von links nach rechts eingetragen werden, und noch einmal dieselben Segmente in die Zeilenanfänge (also die 1. Spalte) fortlaufend von oben nach unten. Die Bewertung zweier Segmente zueinander kann dann an den Schnittpunkten von Spalten und Zeilen eingetragen werden.

Tab. 5.2 Vorschlag für numerische Codes zur Kohärenzbewertung

Kohärenzverhältnis	Code
mit Sicherheit keine kohärenten Segmente	−2
wahrscheinlich keine kohärenten Segmente	−1
wahrscheinlich kohärente Segmente	1
mit Sicherheit kohärente Segmente	2

Für kleinere Transkripte oder Ausschnitte kann diese Matrix direkt während der Bewertung in einer Tabellenkalkulation erstellt werden, aber für größere Segmente wird dies schnell unhandlich. Für die Analysen in Donick (2016) habe ich daher ein einfaches Werkzeug programmiert, das die jeweils zu bewertenden Transkripte importiert, alle Segmente paarweise anzeigt, die Kohärenzbewertung für jedes Paar entgegennimmt und am Ende des Vorgangs die Analysematrix im csv-Format speichert. Die csv-Dateien habe ich in eine Tabellenkalkulation importiert, wo ich unterschiedliche Einfärbungen vornehmen, den Kohärenzverlauf im zeitlichen Verlauf auf einen Blick beurteilen und beliebige Berechnungen durchführen konnte. Selbstverständlich wäre auch die Entwicklung eines Werkzeugs denkbar, das diese Möglichkeiten direkt bietet.

Beachten Sie, dass die Matrixdarstellung nur ein Hilfsmittel ist, auch wenn ich im Rest dieses Abschnitts bei dieser Form bleibe. Es sind auch andere Arbeits- und Darstellungsformen möglich, um Kohärenzbeziehungen festzuhalten. Um beispielsweise meine Bewertungen in Donick 2016 für Leser*innen aufzubereiten, habe ich zu bewertende Segmentpaare direkt am Transkript mit geschweiften Klammern unterschiedlicher Farben verknüpft. Wichtig ist letztlich, dass Sie eine Form finden, die für Sie und ggf. Ihre Kolleg*innen übersichtlich und gut zu handhaben ist.

Beispiel für eine Kohärenzbewertung

Betrachten Sie nun bitte das Transkript in Beispiel 5.6:

Beispiel 5.6

1	Jack:	So, dann klicke ich mal … auf „Speichern“ (… 12s)
2	Jack:	Hm, wieso macht … der jetzt nichts mehr? (… 3s)
3	Jack:	Was? „Die Datei konnte nicht gespeichert werden“? (… 4s)
4	Jack:	„Auf Laufwerk C: ist nicht genug“ … „SPEICHERPLATZ vorhanden“?
5	Jack:	Aber der Stick hat doch genug …
6	Jack:	Das KANN doch gar nicht SEIN!

Sie kennen die gezeigte Sequenz bereits aus Beispiel 5.3. Während wir uns dort einzelne Begriffe innerhalb der Transkriptzeilen angeschaut haben, betrachten wir nun größere Bereiche als Segmente. Wir setzen jedes Segment mit jedem anderen Segment in Beziehung und bewerten diese Beziehung mit den eben eingeführten Codes von −2 bis +2. Für das Segment 1 in Relation zu den Segmenten 2 bis 5 sieht das in einer Matrixdarstellung aus wie in Tab. 5.3.

Tab. 5.3 Segment 1 in Beziehung zu den Segmenten 2 bis 6 (zu Beispiel 5.6)

		1
		So, dann klicke ich mal … auf „Speichern“ (… 12s)
1	So, dann klicke ich mal … auf „Speichern“ (… 12s)	
2	Hm, wieso macht … der jetzt nichts mehr? (… 3s)	2
3	Was? „Die Datei konnte nicht gespeichert werden“? (… 4s)	2
4	„Auf Laufwerk C: ist nicht genug“ … „SPEICHERPLATZ vorhanden“?	1
5	Aber der Stick hat doch genug …	1
6	Das KANN doch gar nicht SEIN!	−2

Wie kommt die Bewertung in Tab. 5.3 zustande? *Tatsächlich entstand sie intuitiv und spontan beim* Lesen der einzelnen Segmente. Einige Relationen habe ich als mit Sicherheit kohärent (2) bewertet, andere als wahrscheinlich kohärent (1), und eine als mit Sicherheit nicht kohärent (−2). Wenn Sie sich an dieses Verfahren gewöhnt haben, werden Sie oft auch so schnell vorgehen. Für die Nachvollziehbarkeit ist es aber sinnvoll, die eigenen Entscheidungen zu reflektieren. Wenn wir dies im Beispiel tun, finden wir zum einen wieder die Verknüpfungsmittel, über die wir in Abschn. 5.3.1.1 gesprochen haben:

- die Wiederaufnahme des Begriffs „Speichern“ in den Segmenten 3 und 4
- die Erwartung von Ursache und Wirkung als ontologisch begründete Kontiguität in Relation zu Segment 2
- eine außersprachliche, kontextuell herstellbare Verknüpfung zu den in Segment 4 und 5 genannten Speichermedien („Laufwerk“, „Stick“) und dem „SPEICHERPLATZ“.

Interessanterweise habe ich Segmente, für die Referenzidentität oder Kontiguität feststellbar ist, intuitiv als mit Sicherheit kohärent bewertet, während ich Segmente, bei denen ich ‚nur‘ außersprachliche Verknüpfungen erkennen kann, intuitiv als nur wahrscheinlich kohärent bewertet habe. Dies zeigt uns, dass sprachliche Mittel der Wiederaufnahme eine große Bedeutung für das Erkennen von Kohärenz haben. Schauen wir uns an, wie es weitergeht (Tab. 5.4):

Zur Begründung:

- Ein ontologisches Kontiguitätsverhältnis im Sinne von Ursache und Wirkung ist zwischen Segment 2 und 3 erkennbar: „macht […] nichts mehr“ als Verbform, die sich auf die scheinbar fehlende Aktivität des Computers bezieht, wird durch die Meldung „konnte nicht gespeichert werden“ rückwirkend erklärt (im Sinne von: ‚Der Computer reagierte zeitweise nicht, weil er den Speicherbefehl nicht ausführen konnte‘). Deshalb erscheint mir diese Relation als mit Sicherheit kohärent.
- Ein ähnliches Verhältnis vermag ich in der Beziehung zu Segment 4 erkennen, wenngleich mir die Stärke der Kohärenz hier etwas geringer erscheint (Segment 4 ist eher eine Erklärung zu Segment 3, als eine Erklärung zu Segment 2, ist aber über Segment 4 trotzdem mit Segment 2 verknüpft).

Tab. 5.4 Segmente 1 und 2 in Beziehung zu den Segmenten 2 bis 6 (zu Beispiel 5.6)

		1	**2**
		So, dann …	Hm, wieso macht … der jetzt nichts mehr? (… 3s)
1	So, dann klicke ich mal … auf „Speichern“ (… 12s)		
2	Hm, wieso macht … der jetzt nichts mehr? (… 3s)	2	
3	Was? „Die Datei konnte nicht gespeichert werden“? (… 4s)	2	2
4	„Auf Laufwerk C: ist nicht genug“ … „SPEICHERPLATZ vorhanden“?	1	1
5	Aber der Stick hat doch genug …	1	−2
6	Das KANN doch gar nicht SEIN!	−2	1

- Den Ausruf „Das KANN doch gar nicht SEIN!“ in Segment 6 beziehe ich intuitiv nicht nur auf die Fehlermeldung des Computers, sondern auf den (wohl als frustrierend erlebten) Vorgang als Ganzen, beginnend mit der scheinbar fehlenden Aktivität zu Beginn, weshalb ich die Beziehung zu Segment 2 immerhin noch als wahrscheinlich kohärent ansehe.
- Mit Sicherheit keine Kohärenz habe ich in der Beziehung zu Segment 5 erkannt.

Auch hier sehen wir also, dass die Bewertung von Kohärenzrelationen in erster Linie intuitiv und spontan geschehen kann, dass sich aber spontane Bewertungen in der Regel begründen lassen. Tab. 5.5 zeigt den vollständigen Bewertungsverlauf, wobei die Darstellung aus Platzgründen komprimiert ist.

Zur Begründung:

Tab. 5.5 Alle Segmente des Beispiels 5.6 in Beziehung zueinander

		1	**2**	**3**	**4**	**5**
		So …	Hm …	Was …	Auf …	Aber …
1	So, dann klicke ich mal … auf „Speichern“ (… 12s)					
2	Hm, wieso macht … der jetzt nichts mehr? (… 3s)	2				
3	Was? „Die Datei konnte nicht gespeichert werden“? (… 4s)	2	2			
4	„Auf Laufwerk C: ist nicht genug“ … „SPEICHERPLATZ vorhanden“?	1	1	2		
5	Aber der Stick hat doch genug …	1	−2	2	2	
6	Das KANN doch gar nicht SEIN!	−2	1	2	2	2

- Die Segmente 3 und 4 sind mit Sicherheit kohärent, weil Segment 4 die kausale Erklärung für die in Segment 3 auftretende Fehlermeldung ist. Zudem sind die Segmente durch Kontiguität („gespeichert“, „SPEICHERPLATZ“) gekennzeichnet.

- Die Segmente 3 und 5 bzw. 4 und 5 sind aufgrund der Konjunktion „aber“ sowie des außersprachlichen Kontexts (Jacks Annahme, er würde auf dem Stick mit ausreichend Speicherplatz speichern) mit Sicherheit kohärent.
- Zwischen den Segmenten 3 einerseits und 4 bis 6 andererseits stellt das Pronomen „das“ eine Beziehung her, die mit Sicherheit kohärent erscheint. Dies wird noch verstärkt durch den außersprachlichen Kontext (der offensichtlichen Frustration, die sich in der lauten Äußerung niederschlägt).

Wird die Analyse der Lokalstruktur direkt in Form der gezeigten Kohärenzanalyse vorgenommen (und werden die sprachlichen Mittel erst bei Bedarf im Nachgang genauer bestimmt), lässt sich viel Analysezeit einsparen. Es ist dabei nicht verwerflich, wenn Sie dabei der Intuition zu vertrauen, die Sie sich als Mitglied einer Sprachgemeinschaft erworben haben. Wenn Sie an größerer Sicherheit interessiert sind, lassen Sie die Bewertung unabhängig von zwei oder drei Personen durchführen und bestimmen Sie danach die Abweichungen (das nennt sich in Anlehnung an traditionelle Gütekriterien Intercoder-Reliabilität).

Wozu brauchen wir das?

Die Kohärenzanalyse mit numerischen Codierungen bietet die Möglichkeit, einen strukturierten Überblick über den Verlauf einer aufgezeichneten Beobachtung zu gewinnen. Wird die Analysematrix farbig dargestellt, erlaubt sie die schnelle Identifikation ‚interessanter‘ Sequenzen, beispielsweise (nach Donick und Tavangarian 2010, S. 281):

- *konstante Themennähe*: positive Relationen in größeren, zusammenhängenden Bereichen – zeigt fortlaufende, wahrscheinlich problemlose Nutzung an; Nutzer*innen und Software beziehen sich aufeinander
- *plötzliche lokale Themenferne*: negative Relationen in ansonsten positiv bewerteten Bereichen – deutet auf unerwartete Irritation während eines ansonsten ‚sauberen‘ Nutzungsprozesses hin
- *konstante Themenferne/‚aneinander vorbei reden‘*: positive Relationen innerhalb der Äußerungen von Nutzer*innen, aber gleichzeitig negative Relationen zu Segmenten, die Benutzeroberfläche oder Aktivität von Software zeigen – deutet darauf hin, dass Nutzer*innen glauben, den Nutzungsprozess zu verstehen, dies aber unter Umständen nicht tun
- *unsaubere Themenwechsel*: negative ‚Ausfransungen‘ an den Rändern positiver Bereiche – kann Unsicherheit bei der Nutzung zeigen, wobei die Nutzung aber nicht abbricht und auch keine konstanten Missverständnisse erkennbar sind.

Insbesondere Themenferne und unsaubere Themenwechsel können für eine eingehendere Betrachtung lohnend sein, zeigen sie doch mögliche Problemstellen der Beziehung zwischen Nutzer*in und Software auf.

Abschließend sei ergänzt, dass hohe Kohärenz allein nicht bedeutet, dass es keine Problembereiche gibt. Im schlimmsten Fall kann ein Transkript ‚lauten Denkens' hohe Kohärenz aufweisen, aber trotzdem keine erfolgreiche Softwarenutzung zeigen. Dies kann bei Nutzungsschwierigkeiten passieren, die Nutzer*innen zwar ausführlich kommentieren, aber nicht zielführend bearbeiten. Dann werden Sie trotzdem Wiederaufnahmen und außersprachliche Verknüpfungen finden (und vielleicht keine Themenferne erkennen). Falls Sie vorhaben, Kohärenzrelationen als Grundlage für quantitative Bewertungen heranzuziehen (vgl. Kap. 6), würden solche ‚irreführend positiven' Fälle das Ergebnis verzerren.

Glücklicherweise werden Sie solche Fälle meist schon bei der Analyse der Globalstruktur erkennen und können sie dementsprechend als Sonderfall berücksichtigen. Eine weitere Möglichkeit ist es (wie schon mehrfach angesprochen), nicht allein die Transkripte ‚lauten Denkens' als Material zu verwenden, sondern beispielsweise Logdateien von Softwareaktivität oder aufgezeichnete Bildschirminhalte hinzuzuziehen – in sich hochkohärente Nutzer*innen-Kommentare wären in Relation zu den Segmenten der Logdateien oder der Bildschirminhalte wahrscheinlich weniger kohärent, wodurch die Themenferne der Nutzer*innen-Kommentare und damit die problematische Nutzungssituation deutlich würde.

5.3.3 Strukturelle Kopplung (Differenz)

Ein rein vom Bewertungsvorgang der Kohärenzanalyse (vgl. Abschn. 5.3.2) sehr ähnlicher, aber in der Bedeutung und Begründung der Bewertungen etwas anderer Weg stellt die Herausarbeitung struktureller Kopplungen dar. Anstatt Kohärenz als linguistischer Kategorie steht nun die systemtheoretisch orientierte Beobachtung von Nutzer*in und Software als System-Umwelt-Beziehung im Vordergrund (vgl. Kap. 3). Es geht darum zu zeigen, dass Nutzer*innen bestimmte *Unterscheidungen* treffen und mit jeder Unterscheidung auch denkbare Alternativen ausschließen. Eine Unterscheidung im systemtheoretischen Sinne ist zuallererst eine Beobachtung des beobachtenden Systems und daher an sich nicht von außen erkennbar. Außenstehende Beobachter*innen zweiter Ordnung (vgl. Kap. 1) können aber anhand sichtbarer Handlungen (zum Beispiel Eingaben am Computer) und kommunikativer Äußerungen (wie beim ‚lauten Denken') auf das Vorhandensein von Unterscheidungsprozessen schließen und die Entwicklung dieser Prozesse im zeitlichen Verlauf nachvollziehen (dies habe ich in Donick 2016 ausführlich gezeigt).

Nutzung als Kopplungsprozess beobachten

Die für die Praxis relevante Idee ist nun, dass sich Nutzungsprobleme zeigen, wenn das Fortschreiten von Unterscheidung zu Unterscheidung gestört wird oder aufhört. Im schlimmsten Fall sind die betreffenden Menschen dann nicht mehr als System „Nutzer*in" zu beobachten, sondern als Teil eines Interaktionssystems mit anderen Menschen (zum Beispiel, wenn sie beim Support anrufen oder Sie um Hilfe bitten).

Anstatt die Beziehung zweier Segmente anhand vorwiegend linguistischer Kohärenzkriterien zu bewerten, stehen daher die folgenden vier Kategorien im Fokus, die wir in Anlehnung an Spencer-Browns *Laws of Form* (2011) verwenden (vgl. Kap. 2; Donick 2016, S. 167 f.):

- *Aufbau einer strukturellen Kopplung*: Ein Segment entwickelt sich mit einem vorher eingeführten Segment zu ‚etwas Neuem' weiter – zum Beispiel führen Eingaben von Nutzer*innen zu darauf bezogenen Ausgaben der Software. Spencer-Brown nennt dies *Kondensierung* (Zusammenziehen) zweier Elemente auf die Unterscheidung, die zwischen ihnen besteht (Spencer-Brown 2011, S. 8).
- *Bestehen eines gekoppelten oder ungekoppelten Zustands*: Ein Segment bestätigt ein vorher eingeführtes Segment, ohne etwas Neues zu ergänzen – zum Beispiel wiederholen Nutzer*innen mehrfach dieselben Eingaben, ohne dass eine Änderung eintritt, oder aufeinanderfolgende Äußerungen von Nutzer*innen sind inhaltlich identisch. Spencer-Brown nennt dies die *Konfirmierung* (Wiederholung) zweier Elemente (ebd.).
- *Aufbruch einer strukturellen Kopplung*: Ein Segment hebt eine vorher gemachte Unterscheidung wieder auf – zum Beispiel unterbrechen Nutzer*innen den Umgang mit der Software, um etwas ganz anderes zu machen, etwa einen Kaffee holen oder sich mit einer anderen Person unterhalten. Spencer-Brown nennt dies die *Aufhebung* zweier Elemente (ebd., 9).
- *Zustand, der den Aufbau einer neuen Kopplung ermöglicht*: Dies ist ein sozusagen undefinierter Zustand, bei dem zurzeit keine Unterscheidungen beobachtbar sind; es liegt keine Kopplung (mehr) vor. Gerade weil dies der Fall ist, sind aber künftig neue Unterscheidungen bzw. der Aufbau neuer Kopplungen möglich – zum Beispiel eine bewusst eingelegte Pause bei der Softwarenutzung, bei der ein beobachteter Mensch nicht im Modus ‚Nutzer*in' zu beobachten ist, von dem aber anzunehmen ist, dass das demnächst wieder der Fall ist. Spencer-Brown spricht hier von *Kompensation* (ebd.).

Wie schon gesagt entspricht der reine Ablauf der Bewertung dem der Kohärenzanalyse aus dem letzten Abschnitt. Allerdings werden andere Codes verwendet, um die genannten vier Kategorien besser abzubilden (Tab. 5.6).

Tab. 5.6 Numerische Codes nach Donick (2016)

Kategorie	Code	Bezeichnung nach Spencer-Brown (2011)
Aufbau oder Weiterentwicklung einer strukturellen Kopplung	2	Kondensierung
Bestehen eines gekoppelten oder ungekoppelten Zustands	1	Konfirmierung
Aufbruch einer strukturellen Kopplung	−1	Aufhebung
Zustand, der den Aufbau einer neuen Kopplung ermöglicht	0	Kompensation

Wie erkennen wir nun, welche Beziehung zwischen zwei Segmenten besteht? Woran sehen wir, ob zwei Segmente kondensieren, ob sie sich konfirmieren, ob sie sich aufheben, oder keine Beziehung besteht, also Kompensation vorliegt? Dabei hilft uns wieder die Sprache. Ich habe das einmal wie folgt zusammengefasst: „So sind Aufhebungen etwa an Negationen und Widersprüchen erkennbar. Konfirmierungen und Kondensierungen zeigen sich z. B. in Wiederholungen, Synonymen, Umformulierungen u. ä., wobei bei Kondensierungen zur reinen Wiederholung oft noch etwas Neues hinzukommt. Kompensationen liegen häufig dann vor, wenn die Beziehung nicht als eine der drei anderen Arten erkannt wurde – sondern z. B. als neues Thema“ (Donick 2016, S. 167 f.).

Vertrauen Sie zunächst wieder auf Ihr Sprachgefühl. Während der Bewertung der Segmente eines Transkripts könnten Sie zum Beispiel folgende Eindrücke haben:

- ‚Hier wird auf etwas Vorangegangenem aufgebaut‘ (Kondensierung, 2)
- ‚Das wurde eben schon gemacht‘ (Konfirmierung, 1)
- ‚Das ist doch das Gegenteil von dem, was vorher war‘ (Aufhebung, −1)
- ‚Hier passiert gar nichts mehr‘ (Kompensation, 0).

Bewertungsbeispiel

Zur Verdeutlichung schauen Sie sich bitte das folgende Beispiel (5.7) an.

Beispiel 5.7

1	Jack:	So, dann klicke ich mal … auf „Speichern“ .
2	Jack:	Okay, da ist … da gebe ich den Namen ein …
3	Jack:	Und okay.
4	Jack:	Fertig
5	Jack:	Jetzt erst mal ein Kaffee

In dem Beispiel ist – auch ohne aufwändige Analyse – offensichtlich Softwarenutzung erkennbar: Jack speichert etwas unter einem Namen ab (Zeilen 1 bis 3). Er kommentiert den Vorgang mehrfach als beendet (Zeilen 3 und 4). Abschließend drückt er seinen Wunsch nach einem Kaffee aus (Zeile 5).

Erkennen Sie im Beispiel Kondensierungen, Konfirmierungen, Aufhebungen und Kompensationen?

Hier ist mein Bewertungsvorschlag:

- Die Beziehung der Zeilen 1 und 2, 2 und 3, sowie 1 und 3 ist jeweils als Kondensierung zu bewerten. Wir beobachten hier den fortgesetzten Aufbau einer strukturellen Kopplung. Jack klickt auf „Speichern“, gibt einen Dateinamen ein und klickt auf „Okay“. In jeder Zeile kommt ein weiterer Nutzungsschritt hinzu: etwas Neues, das sich aber logisch aus dem vorangegangenen herleitet.

- Die Beziehung der Zeile 4 zu den Zeilen 1 bis 3 ist als Konfirmierung zu bewerten. Das „Fertig“ bestätigt die Beendigung des bis dahin beobachteten Speichervorgangs, aber es fügt diesem Vorgang nichts Neues hinzu.
- Die Beziehung der Zeilen 4 und 5 wäre unter einem anderen Beobachtungsfokus zwar durchaus als Kondensierung zu werten (der Wunsch nach dem Kaffee ist ableitbar aus dem Beenden der Arbeit), aber nicht, wenn wir an dem Bestehen von Computernutzung als struktureller Kopplung interessiert sind. Hier ist es plausibler, die Beziehung der Zeilen 4 und 5 als Aufhebung des (bis Zeile 4 immerhin noch konfirmierten) Nutzungsvorgangs zu bewerten.
- Die Beziehung der Zeile 5 zum Rest des Transkripts ist als Kompensation zu bewerten. Es liegt kein gekoppelter Zustand mehr vor, es ist aber möglich, dass künftig neue Kopplungen aufgebaut werden.

In Form einer Analysematrix sieht die Bewertung aus wie in Tab. 5.7.

Wozu brauchen wir das?

Anstatt einer ganzen Reihe sprachlicher Marker wie bei der Analyse thematischer Progression (Abschn. 5.3.1) oder den oft unvermeidbaren Spekulationen, die für eine Zuschreibung von vorhandener oder fehlender Kohärenz nötig sind (Abschn. 5.3.2), arbeitet die Analyse struktureller Kopplungsprozesse mit vier ganz grundlegenden, abstrakten Kategorien. Gerade wenn man Nutzer*innen im Zusammenspiel mit Software beobachtet, ist es einfacher, zwei Segmente als Weiterentwicklung, als Wiederholung, als Widerspruch oder als gar nicht zugehörig zu identifizieren, als an Details dieser Strukturen konkrete Themen herauszuarbeiten oder sich in Bezug auf vermutete Kohärenz festzulegen.

Letzteres unterstellt letztlich den Nutzer*innen, dass sie Zusammenhänge sehen würden, obwohl sie das vielleicht gar nicht tun, und das erzeugt eine gewisse Unschärfe in den zu erwartenden Ergebnissen. Solche Unterstellungen werden vermieden, wenn man sich während der strukturellen Analyse konsequent als Beobachter*in zweiter Ordnung versteht und auf Spekulationen verzichtet, die über die Strukturen hinausgehen. An dieser Stelle liegt das Potenzial einer Analyse, die nur die von anderen Systemen hinterlassenen Spuren von Unterscheidungsprozessen beobachtet.

Tab. 5.7 Alle Segmente des Beispiels 6.7 in Beziehung zueinander

		1	**2**	**3**	**4**
		So, dann klicke ich …	Okay, da ist …	Und okay.	Fertig
1	So, dann klicke ich mal … auf „Speichern“ .				
2	Okay, da ist … da gebe ich den Namen ein …	2			
3	Und okay .	2	2		
4	Fertig	1	1	1	
5	Jetzt erst mal ein Kaffee	0	0	0	−1

Literatur

Brinker, Klaus/Sager, Sven F.: Linguistische Gesprächsanalyse. Eine Einführung. Berlin 2006.

Brinker, Klaus: Linguistische Textanalyse. Eine Einführung in Grundbegriffe und Methoden. Berlin 2005.

Donick, Mario: „Offensichtlich weigert sich Facebook, mir darauf eine Antwort zu geben“: Strukturelle Analysen und sinnfunktionale Interpretationen zu Unsicherheit und Ordnung der Computernutzung. Hamburg 2016.

Donick, Mario/Tavangarian, Djamshid: Qualitatives Evaluationsverfahren für interkulturelle E-Learning-Szenarien am Beispiel des Online-M.Sc. „Visual Computing“. In: Apostolopoulus, Nicolas/Mußmann, Ulrike/Rebensburg, Klaus/Schwill, Andreas/Wulschke, Franziska (Hrsg.): Grundfragen Multimedialen Lehrens und Lernens. E-Kooperationen und E-Praxis. Tagungsband GML² 2010, 11.–12. März 2010, Berlin 2010, S. 273–285.

Luhmann, Niklas: Vertrauen. Stuttgart 2009.

Linke, Angelika / Nussbaumer, Markus / Portmann, Paul R.: Studienbuch Linguistik. Tübingen 2004.

Spencer-Brown, George: Laws of Form. Leipzig 2011.

Sucharowski, Wolfgang/Schwennigcke, Bastian: Partizipation. Lesen im sozialen Raum. Rostock 2009.

Analyseergebnisse interpretieren: Software als Medium und Schnittstelle, Quality of Interaction und Gestaltungsnormen

6

Inhaltsverzeichnis

6.1 Vorbemerkung: Was heißt „Interpretation“?

Dieses Buch zeigt Ihnen, wie Sie Menschen bei der Nutzung von Software so beobachten können, dass Sie mit Hilfe der Beobachtungen bestimmte Fragen zur Nutzungssituation und zum Umgang von Nutzer*innen mit der Software beantworten können. Dabei sollen am Ende praktisch verwertbare Ergebnisse stehen, mit denen Sie Software situationsgerechter entwickeln können. In Kap. 4 haben Sie gelernt, wie Sie solche Beobachtungssituationen planen und durchführen. In Kap. 5 haben Sie gelernt, wie Sie die erhobenen Daten strukturell analysieren können – bezüglich Themen, Kohärenz und struktureller Kopplungen. Nun lernen Sie, wie Sie die Analysen entsprechend Ihrer Fragestellungen interpretieren können, um nutzbare Ergebnisse abzuleiten. Der Weg von der Analyse zu verschiedenen Arten der Interpretation war bereits in Kap. 4 (Abb. 4.1) dargestellt; wir gehen auf die darin genannten Interpretationsweisen in diesem Kapitel näher ein. Vorher sollten wir jedoch kurz klären, was ‚interpretieren‘ überhaupt heißt.

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8_6

In Kap. 4 wurde bereits angesprochen, dass das Beobachtungsverfahren in diesem Buch ein qualitatives interpretatives Verfahren ist und dass es sich von quantitativen, ‚härter' natur- und ingenieurwissenschaftlichen Ansätzen unterscheidet (etwa in Bezug auf Gütekriterien, geeignete Fragestellungen und Hypothesenbildung). Es ersetzt quantitative Studien nicht, sondern erlaubt gleichsam einen Blick in die Tiefe der einzelnen Fälle. Dabei können Details zutage treten, die sonst übersehen würden. In Kap. 5 haben Sie wahrscheinlich beides bemerkt: Wir haben uns sprachliche Feinheiten anhand kurzer Beispiele angeschaut und dabei Nutzungsprobleme belegen können. Wir haben auch bemerkt, dass bereits der Umgang mit dem sprachlichen Material eine Interpretationsleistung ist. Sogar zu erkennen, ob ein einzelnes Wort ein anderes Wort wieder aufnimmt (dass oder ob ein Pronomen etwa auf ein vorher eingeführtes Substantiv verweist), verlangt bereits Interpretation vor dem Hintergrund des Sprachverstehens und des Kontexts. Noch mehr Interpretation findet statt, wenn wir Kohärenzbeziehungen aufdecken wollen. Eine gewisse Unschärfe ist wegen der „Vagheit der Alltagssprache" (Brinker und Sager 2006, S. 122) bzw. der „Unwahrscheinlichkeit der Kommunikation" (vgl. Luhmann 1981) schon hier unvermeidbar. Gleichwohl steht am Ende der strukturellen Analyse ein Ergebnis, das sich noch eng ans ‚objektiv' vorhandene Aufzeichnungsmaterial hält. Um nun aber die Bedeutung dieses Ergebnisses für die Fragestellung der Untersuchung zu ermitteln (vielleicht vor dem Hintergrund sensibilisierender Konzepte oder gar für die Beurteilung von Hypothesen), ist wiederum Interpretation nötig.

Hier ist zunächst ein Grundsatz interessant, den Brinker/Sager formulieren: „Der Analytiker […] muß sich auf die gleichen Interpretationsverfahren, Normen, Obligationen und Relevanzsetzungen verpflichten wie die von ihm analysierten Personen, will er nicht vollkommen an dem Sinn des Geschehens vorbeigehen" (ebd., S. 125). Interpretation der Analyseergebnisse meint mit diesem Grundsatz also, sich dem *Sinn* der beobachteten Situation anzunähern. Etwas wird für Menschen sinnhaft, wenn sie diesen Gegenstand nicht nur erleben, sondern sich ihm absichtsvoll (intentional) zuwenden (ebd., S. 129). Die besondere Herausforderung für die Interpretation ist dabei, dass der Sinn damit nicht einfach dasselbe ist wie etwa die strukturell ermittelten Themenbereiche, Kohärenzbeziehungen oder Kopplungsverläufe. Diese strukturellen Elemente sind lediglich Spuren, die auf Sinnkonstruktionsprozesse bei den beobachteten Personen hindeuten. Die Interpretation dieser Spuren auf einen möglichen ‚Sinn' hin verlangt, dass wir einen ähnlichen Hintergrund wie die beobachteten Personen haben bzw. während der Interpretation entwickeln (ebd., S. 126) – was mit Brinker/Sager bedeuten kann, den beobachteten Personen „Bewußtseinsinhalte hypothetisch [zu] unterstellen" (ebd., S. 131). Akzeptiert man diese Unschärfe, kann man *mögliche* Sinnkonstruktionsprozesse für die einzelne beobachtete Situation sehr tief gehend nachzeichnen. Für die Kommunikationsforschung als solche ist das ein legitimes Forschungsziel (ebd., S. 127). Für praktisch verwertbare Ergebnisse erscheint diese Art der ‚Sinnsuche' aber zu spekulativ und würde auch vom Interpretationsaufwand zu weit führen.

Ich schlage daher vor, einen Sinnbegriff zu verfolgen, der sich aus der Systemtheorie herleitet und der sich schon in Kap. 3 angedeutet hat. Dabei geht es um das Nachvollziehen

von Spuren, die darauf hindeuten, dass ein von uns beobachtetes System vor dem Hintergrund seiner Umwelt Unterscheidungen vornimmt, das System also das aktuell Vorhandene vom potenziell Möglichen unterscheidet (Donick 2016, S. 83; Luhmann 1992, S. 104) bzw. es dazu in Beziehung setzt. Wenn beispielsweise ein*e Nutzer*in den Text einer Fehlermeldung liest, fragen wir uns in der Interpretation nicht, ob oder wie der Sinn der Fehlermeldung verstanden wird, sondern wir wollen wissen, ob das Lesen der Fehlermeldung zu weiteren Unterscheidungen führt. Über Ersteres könnten wir nur spekulieren, wenn wir der Person Bewusstseinsinhalte unterstellten, die wir selbst als ‚hoffentlich sinnvoll' ansehen. Aber Letzteres können wir direkt am weiteren Beobachtungsverlauf erkennen. Die strukturelle Analyse als Herausarbeiten von Kopplungsprozessen (vgl. Abschn. 5.3) ist dafür eine geeignete Grundlage – und die Interpretation kann damit sogar enden, wenn wir allein an der fortgesetzten Nutzung als solcher interessiert sind, ohne weitere Fragen zu haben.

Mitunter interessieren wir uns jedoch für Aspekte, die durch das Nachvollziehen bloßer Nutzung nicht zu beantworten sind. In Donick (2016) interpretierte ich Transkripte beispielsweise dahingehend, ob sich die Software für eine bestimmte Aufgabe eignete, ob Nutzer*innen der Software vertrauten, oder ob die Nutzung bestimmten Vorgaben und Normen genügte. Zwei dieser Fälle möchte ich im Folgenden darstellen, um Ihnen die Möglichkeiten weitergehender Interpretation zu zeigen (Abschn. 6.2). Anschließend schauen wir uns an, was wir aus Analysen und Interpretationen über Softwarequalität sagen können (Abschn. 6.3).

6.2 Interpretationsbeispiele

6.2.1 Software als Medium

Wenn Nutzer*innen eine Software verwenden und dabei das Vorhandensein struktureller Kopplungen zu beobachten ist, dann kann man diese Software als *Medium* bezeichnen. In systemtheoretischen Begriffen ist etwas dann ein Medium, wenn es ein anderes System zu einer Unterscheidung (= Selektion) motiviert (vgl. Baecker 2007, S. 179). Die Eigenschaft des Mediums, zu Unterscheidungen zu motivieren, bezeichnet Dirk Baecker als Selektivität (ebd., S. 179) des Mediums. Sie ist besonders prägnant auf der Form-Seite des Designs des Mediums (vgl. Abschn. 3.2.2) zu beobachten, zum Beispiel der Benutzeroberfläche der Software.

Es ist wichtig, Selektion hier nicht technisch zu verstehen (etwa als Mausklick in einem Menü). Schon die bewusste Hinwendung zu einer wahrgenommenen Sache ist eine Selektion oder Unterscheidung: Die Sache wird vor einem Hintergrund vieler weiterer potenziell relevanter Dinge ausgewählt oder eben unterschieden. Dieses Konzept beruht auf Luhmanns Verständnis von Kommunikation, der als dreifache Differenz aus Mitteilung, Verstehen und Information angelegt ist (Luhmann 1992, S. 25). Man kann eine Mitteilung auf unterschiedliche Arten verstehen. Verstehen heißt mit Luhmann, Unterscheidungen zu

treffen, um so weitere kommunikative Anschlüsse zu ermöglichen. Kommt es zu solchen Selektionen, wurde die Mitteilung verstanden – und zwar erst mal ganz unabhängig davon, ob das erlangte Verständnis etwa ‚richtig' oder ‚falsch' war. Sie ergibt Sinn, weil sie zu weiteren Anschlüssen motiviert. Insofern „nur ein Nebeneffekt" (ebd., S. 27) ist dabei ein Informationsgehalt, den man damit vielleicht noch verbindet.

Auf Softwarenutzung angewandt meint dies, dass Software Medium ist, wenn eine Eigenschaft der Software von eine*r Nutzer*in wahrgenommen und in irgendeiner Weise verarbeitet wird. Als Außenstehende, die wir Nutzer*innen beobachten, können wir Spuren der Unterscheidungen, die diese Nutzer*innen treffen, zum Beispiel an Eingaben beobachten, die eine Anschlusshandlung zeigen, an nutzungsbegleitenden Kommentaren, oder daran, dass die Nutzung beendet wird. Für die praktische Arbeit wird dies vor allem auf zwei Weisen relevant:

- Welche Elemente einer Benutzeroberfläche tragen zur Selektivität der Software für die Nutzer*innen in der Nutzungssituationen bei (und welche nicht)?
- Sind die Selektionen der Nutzer*innen solche, die aus Entwickler*innen-Sicht für die Nutzungssituation erwartet wurden (oder ganz andere oder ‚falsche')?

Um ein ganz einfaches Beispiel zu geben: Sie platzieren ein Icon in einer Symbolleiste, mit dem man eine bestimmte Funktion ausführen kann. Das Icon ist erst mal nur da; es ist nur ein weiteres Element einer komplexen Umwelt, die vom beobachtenden System auf ein bearbeitbares Maß reduziert werden muss. Das Icon wird erst sinnvoll, wenn ihm jemand Selektivität zuschreibt, das heißt: das Icon wahrnimmt (Mitteilung), es als bedienbar erkennt und bedient (Verstehen) und es somit einer Funktionalität zuordnet (Information). Dann wird das Icon zum Medium (und nun übrigens auch in der Alltagsbedeutung des Begriffs, nämlich als Vermittler zwischen der Absicht der Nutzer*innen und dem Funktionsumfang der Software).

Für eine am Medium interessierte Interpretation der strukturellen Analyse sind nun solche Themen, Kohärenzbereiche oder Kopplungsverläufe interessant, die auf vorhandene oder fehlende Selektivität hinweisen. Sie identifizieren Elemente, die Selektionen motivieren, und Elemente, die auf stattgefundene Selektionen hinweisen. Trotz der abstrakten Begrifflichkeiten ist das relativ einfach zu erkennbar – im ‚Lauten Denken' weisen Nutzer*innen aktiv darauf hin, was sie gerade wahrnehmen und benutzen. Im einfachsten Fall kann das aussehen wie in dem (fiktiven) Beispiel 6.1:

Beispiel 6.1

1	Jack	Der fragt jetzt … „Wollen Sie die Seite wirklich verlassen?" (… 4s)
2	Jack	Nein … ich klick mal „Auf der Seite bleiben"

Zu Beginn nimmt Jack ein Element der Benutzeroberfläche der verwendeten Software wahr; es handelt sich um eine Frage, auf die er mit einer Eingabe antworten kann (Segment 1). Da Jack die Frage vorliest, haben wir darin eine deutliche Spur dafür, dass hier

eine Unterscheidung stattgefunden hat: Jack hat die Frage der Software und die Antwortmöglichkeiten vor dem Hintergrund der anderen sichtbaren Elemente unterschieden bzw. als relevant selektiert. Diese Selektion motiviert Jack nun ihrerseits zu einer weiteren Selektion, nämlich zur Unterscheidung der möglichen Antwort-Alternativen auf die Frage der Software und zur Entscheidung für eine der Alternativen (Segment 2). Die Spur, an der wir das Vorhandensein dieser Selektion ablesen können, ist der von Jack kommentierte Mausklick. Das beobachtbare Oszillieren zwischen Motivation und Selektion zeigt uns, dass hier Software als Medium funktioniert.

Schauen wir uns nun den Ausschnitt einer ‚echten' Nutzungssituation an. In dem Transkriptausschnitt in Beispiel 6.2 wird die Medialität von Software an mehreren Stellen deutlich. Das Beispiel entstammt einem Transkriptkorpus, das ich in Donick (2016) verwendet habe. Es wurde im Jahr 2012 aufgezeichnet. Es zeigt eine Nutzerin, die ihr Konto beim Social Network *Facebook* löschen will. Da die Nutzerin nicht darauf vertraut, dass beim Entfernen des Kontos auch alle von ihr hochgeladenen Fotos entfernt werden, will sie die Fotos vor der eigentlichen Kontoschließung selbst löschen. Ich durfte sie in der Nutzungssituation beobachten und ihre Äußerungen (ihr ‚Lautes Denken') aufzeichnen (vgl. Donick 2016, S. 222–243, 2019, S. 76–83).

Beispiel 6.2: Medium als Selektivität (aus dem Korpus von Donick 2016 und Donick 2019)

[….]
57 [gepresst] Ob ich dann vielleicht vorher erst mal alle meine F … Fotos lösche?
58 Hm keine Ahnung ob das was bringt
[…]
62 FOTOS … [tiefes Einatmen und Seufzen]
63 Mhm, wer is denn hier im Chat? (… 4s)
64 so ich will die alle löschen [räuspern]
65 ah hier
66 Oh Gott [matt]
67 Offensichtlich kann man die nicht einfach so löschen
68 scheiße …
69 kann ich vielleicht die Alben löschen? (… 6s)
70 Hier unter „Bearbeiten" „Album löschen"
71 JA Album löschen …
72 So dann mach ich das Gleiche jetzt mit dem nächsten unbenannten Album …
73 „Bearbeiten" „Album löschen"
74 Ja Album löschen …
75 Gut dann mach ich den [ganzen?] [Müll?] hier [auch?]
76 Uhmm – kann ich das nicht LÖSCHEN? [stark fragend]
77 SCHWEINEBACKEN! (… 6s)
78 „Optionen" „Dieses Foto löschen"
79 ja …

80 „Optionen“ „Dieses Foto löschen“
81 „Bestätigen“
[…]

1. An welchen Stellen des Transkripts können Sie Software als Medium beobachten?
2. Können Sie dafür auch strukturelle Belege anführen (Wiederaufnahmen, Kohärenz, Kopplungen)?

In dem folgenden Interpretationsvorschlag wähle ich die meiner Ansicht nach prägnantesten Segmente aus und kommentiere ihre Beziehung hinsichtlich der Medialität. Strukturelle Belege für die Interpretation führe ich direkt danach an, wobei ich, wenn möglich, Ergebnisse aus allen drei strukturellen Analyseformen nenne, also Wiederaufnahmen, Kohärenzbeziehungen und Kopplungsverläufe. Meine Darstellung verbindet Analyse und Interpretation in bewusst verdichteter Form. Ich will Ihnen damit demonstrieren, in welch kurzer Form sich eine doch relativ lange Sequenz analysieren und interpretieren lässt. Der Nachvollzug meines Vorschlags setzt voraus, dass Sie die in Kap. 5 vorgestellten Verfahren kennen; wenn Sie das Kapitel also noch nicht gelesen haben, holen Sie das bitte nach und versuchen Sie, die Vorschläge auch am Transkript nachzuvollziehen.

- Segment 57 und 62: Die Nutzerin ruft die Fotos ihres *Facebook*-Kontos auf. Sie hat bereits vor der Nutzung eine Unterscheidung getroffen, nämlich die, dass sie *Facebook* nicht traut und darum vor dem Entfernen ihres Kontos alle hochgeladenen Fotos löschen will. Dies kann für die ganze Sequenz als Grundunterscheidung festgehalten werden. Ohne diese Selektion käme es nicht zu der beobachteten Nutzung. Strukturell ist die Unterscheidung an der Wiederaufnahme des Substantivs „Fotos“, an vorhandener Kohärenz zwischen den Segmenten und einer Kondensierung beider Segmente erkennbar.
- Segment 65: Es wurde offensichtlich ein Element der Benutzeroberfläche unterschieden, dass dem Ziel der Nutzerin dienlich zu sein scheint. Strukturell ist die Unterscheidung erkennbar an der Äußerung „ah hier“. Die Äußerung stellt Kontiguität zur vorher eingeführten Idee des Löschens her. Entsprechend besteht auch Kohärenz zu den vorigen Segmenten und es liegt eine Konfirmierung des vorher gekoppelten Zustands vor.
- Die Segmente 66 bis 68 zeigen dann aber, dass das Löschen wohl doch nicht so einfach ist. Zwar ist strukturell weiterhin Kohärenz zum vorher Eingeführten vorhanden, aber die entsteht vorwiegend durch die Negation „nicht“. Daher ist hier die Aufhebung des bis dahin gekoppelten Zustands zu erkennen.
- Segment 67 und 69: Die Nutzerin hat also vor den anderen bis dahin unterschiedenen Elementen nun unterschieden, dass ein Löschen aller Fotos auf einmal nicht funktioniert.

Dies motiviert zu einer neuen Selektion: Statt aller Fotos auf einmal will die Nutzerin nun Fotoalben löschen. Strukturell ist hier Kontiguität als Teil-Ganzes-Beziehung erkennbar (‚Fotoalben' sind nicht derselbe außersprachliche Gegenstand wie ‚alle Fotos', aber Teil dieser Gesamtmenge), auch Kohärenz zu vorher eingeführten Segmenten (es geht immer noch um das Löschen, das auch sprachlich wieder aufgenommen wird). Der ungekoppelte Zustand wird in diesen Segmenten kompensiert und ermöglicht in der Folge neue Unterscheidungen.

- Segment 70 und 71 sowie 73 und 74: Die Nutzerin unterscheidet Elemente der Benutzeroberfläche („Bearbeiten"), die zu weiteren Elementen der Oberfläche führen („Album löschen"). Diese motivieren dazu, sie zu verwenden, was die Nutzerin im Kommentar bekräftigt („Ja Album löschen"). Strukturell ist dies an Wiederaufnahmen, Kohärenz und Kondensierungen nachweisbar.
- Segment 76 und 78: Aus unbekannten Gründen scheint die Selektion in Segment 74/76 nicht mehr brauchbar zu sein; die Nutzerin kann auf einmal keine Alben mehr löschen, was eben noch ging. Die bis hierher bestehende Kopplung wird aufgehoben; wieder erkennbar an der Negation „nicht" und der Wiederaufnahme. Nach einer kurzen Kompensation (die in Segment 77 auch strukturell als Schimpfwort erkennbar ist) motiviert dies die Selektion, nun jedes Foto einzeln zu löschen (was zum vorher eingeführten wieder Kontiguität als Teil-Ganzes-Beziehung bildet: das einzelne löschbare Foto als Teil des nicht löschbaren Albums).
- Segment 78 und 79, sowie 80 und 81: Die Nutzerin unterscheidet Elemente der Benutzeroberfläche („Optionen"), die zu weiteren Elementen der Oberfläche führen („Dieses Foto löschen"). Diese motivieren wieder dazu, sie zu verwenden, was die Nutzerin wiederum mit einem Kommentar bestätigt („ja" und „Bestätigen"). Strukturell ist dies an Wiederaufnahmen, Kohärenz und Kondensierungen nachweisbar.

Zusammengefasst zeigt sich die Software wieder an den Stellen als Medium, an denen das Wechselspiel aus Motivation und Selektion zu beobachten ist, wobei Motivationen und Selektionen über die sichtbare Software hinausgehen können.

Was die gezeigte Sequenz (im Gegensatz etwa zu Beispiel 6.1) so problematisch macht, sind die Inkonsistenzen im medialen Charakter der Software. Die Software kann zwar insgesamt als Medium beobachtet werden, da sie zu Selektionen motiviert, doch sind die motivierten Selektionen nur teilweise zielführend. Sie sind zielführend, wo dem Ziel der Nutzerin entsprechende Eingriffsmöglichkeiten („Bearbeiten", „Album löschen", „Optionen", „Dieses Foto löschen") angeboten werden. An anderen Stellen jedoch wird Selektivität versprochen (Segmente 65, 70, 73), die gar nicht oder auf einmal nicht mehr eingelöst wird (Segmente 67, 76). Dadurch ist die Nutzerin mehrfach gezwungen, den einmal begonnenen Ablauf zu unterbrechen und Alternativen zu finden, um ihr Ziel zu erreichen. Eine effiziente Nutzung ist so nicht möglich. Entwickler*innen müssten an der Stelle eigentlich eingreifen und prüfen, ob die Abläufe konsistenter zu gestalten wären.

6.2.2 Software als Schnittstelle

Wenn Selektivität eines Mediums nur oder vorwiegend zu Selektionen führt, die wenig zielführend sind, ist der Schnittstellencharakter des Mediums gefährdet. Mit *Schnittstelle* meint die Systemtheorie nicht die Benutzeroberfläche (das Interface) einer Software, sondern Mittel und Verfahren, um bei Konflikten zu intervenieren (Baecker 2007, S. 276). In unserem Zusammenhang kann der Konflikt als das Nutzungsziel betrachtet werden, weswegen wir eine Software einsetzen. Das Nutzungsziel ist ein Problem und die Software verwenden wir zur Problemlösung. Mittels der Software interveniert ein*e Nutzer*in hinsichtlich des Ziels, das erreicht werden soll (Donick 2016, S. 213): „Die Benutzeroberfläche ist nur eine kommunikative Struktur, die für einen Beobachter auf die dahinter liegende Schnittstellenfunktion verweist“ (ebd.). Diese Erkenntnis steckt auch im bekannten Designgrundsatz ‚form follows function‘. Software als Schnittstelle verweist also auf die „Funktion“-Seite ihres Designs (vgl. Abschn. 3.2.2), sowie auf das Verhältnis von Plan und Abweichung (vgl. Abschn. 3.2.3).

Das Ziel der Nutzung kann weit über die eigentliche Nutzungssituation hinausgehen. Im Beispiel 6.2 aus dem letzten Abschnitt bestand die konkrete Absicht zwar darin, Fotos von einer Online-Plattform zu löschen. Dabei war die Online-Plattform selbst auch die Software, die zum Löschen der Fotos eingesetzt wurde. Ob wir diese Software aber als geeignete Schnittstelle beurteilen können, hängt von dem Ziel ab, das hinter dem Löschen der Fotos stand – dem Konflikt, in den mit der Software interveniert werden sollte:

- Ging es der Nutzerin nur darum, ein bisschen Ordnung unter den hochgeladenen Fotos zu schaffen oder Speicherplatz zu sparen?
- Kuratierte sie die hochgeladenen Fotos, um sich in bestimmter Weise als Person zu inszenieren (vgl. Krotz 2007, S. 108; Donick 2016, S. 131ff.)?
- War das Löschen der Fotos Symptom einer Vertrauenskrise, das die Plattform insgesamt betraf?

In den ersten beiden Fällen wäre die Software durchaus als geeignete Schnittstelle zu betrachten – zwar nicht als besonders effiziente, aber im Großen und Ganzen doch funktionierende. Doch im letzten Fall (und das ist der, der in der aufgezeichneten Situation vorlag) ist das zweifelhaft. Der Konflikt besteht darin in zweifacher Hinsicht:

- In erster Ordnung liegt der Konflikt darin, dass die Nutzerin der Plattform nicht mehr genug traute, um dort weiter Mitglied sein zu wollen. Das Misstrauen ging so weit, dass angenommen wurde, dass das Löschen ihres Kontos möglicherweise nicht dazu führen würde, dass auch ihre hochgeladenen Fotos gelöscht würden. Die in Beispiel 6.2 besprochene Sequenz zeigt den Versuch, das Löschen der Fotos durch eigenes Handeln doch irgendwie sicherzustellen. Ausgerechnet das aber war auch nur mit Unterbrechungen und in wenig konsistenter Weise möglich.

- In zweiter Ordnung bestand der Konflikt in der Schwierigkeit, in Bezug auf die Nutzung der Plattform Selbstwirksamkeit zu erleben, also die eigene Person als wirklich selbstbestimmt und das eigene Handeln als erfolgreich wahrzunehmen. Statt die Nutzerin dabei zu unterstützen und die Plattform dadurch erst als vertrauenswürdig zu präsentieren, legte die Plattform der Nutzerin scheinbar absichtlich Steine in den Weg.

Es ist hier zweitrangig, wie es überhaupt zu diesem zweifachen Konflikt kam, ob durch negative Medienberichte über die Plattform, durch Hörensagen von Freunden und Bekannten, oder durch eigene negative Erfahrung. Relevant ist, dass kein Vertrauen in die Plattform und deren Umgang mit hochgeladenen Fotos vorlag, und dass die Nutzerin dieser Plattform relativ machtlos gegenüberstand. Das umständliche Löschen jedes einzelnen Bildes nacheinander wirkt als fast verzweifelter Selbstbehauptungsversuch, die Hoheit über die eigenen Daten zu behalten, in einer Umgebung, die sich nicht gerade interessiert an diesem Versuch zeigte. Ob das an bloß nachlässiger, vielleicht nicht ganz durchdachter Konzeption der Plattform und ihrer Benutzeroberfläche lag, oder ob Motive dahinter steckten, die eher dem Wirtschaftsunternehmen hinter der Plattform zuzuschreiben sind, ist nicht entscheidend; wichtig ist, dass sich die Software in der Nutzungssituation nicht als geeignete Schnittstelle präsentiert hat und damit das ohnehin schon gefährdete Vertrauen noch weiter zerstört hat (vgl. zum Vertrauen in Technik ausführlich Donick 2019).

Die Schnittstellen-Problematik des Beispiels 6.2 kann für andere Anwendungsbereiche ähnlich beschrieben werden. Hier drei Beispiele:

- In Donick 2016 interpretiere ich ein Let's-Play-Video, in dem ein Spieler im Strategiespiel *Europa Universalis IV* die Rolle eines historischen Herrschers einnimmt (Donick 2016, S. 212–217). Das Spiel war Schnittstelle, um die mit der Rolle verbundenen Spielziele zu bewältigen (obwohl sich dabei im Einzelnen einige Probleme zeigten).
- Das Buch, das Sie gerade lesen, und die darin beschriebenen Verfahren und Anregungen sind im besten Fall Technik (vgl. Kap. 1), die Ihnen bei der Intervention in Ihrem Arbeitsfeld hilft. Ob das Buch dafür aber wirklich eine geeignete Schnittstelle ist, zeigt sich erst vor dem Hintergrund Ihres persönlichen Kontexts und wenn Sie die beschriebenen Verfahren in tatsächlichen Situationen anwenden (also zum Beispiel in verschiedenen Entwicklungsphasen einer Software, an der Sie gerade arbeiten, ausprobieren).
- Die Textverarbeitung, mit der ich das vorliegende Buch geschrieben habe, zeigt sich als Schnittstelle, um nicht nur den Konflikt ‚Manuskript rechtzeitig fertigstellen' zu bearbeiten, sondern darüber hinaus auch, mich selbst in der Rolle eines Autors von Sach- und Fachbüchern zu inszenieren.

In allen drei Beispielen geht der Schnittstellencharakter der jeweiligen Technik über die Nutzungssituation im engeren Sinne hinaus: Im Spiel geht es nicht nur die Bewältigung der Spielmechanik, sondern um das Spiel der Rolle eines europäischen Herrschers. Im Buch geht es nicht nur um das Lesen der einzelnen Kapitel, sondern um den praktischen Nutzen, den Sie davon hoffentlich haben. Und im Textprogramm geht es nicht nur

um das Schreiben eines Buches, sondern um den Beitrag, den dieses Schreiben für die berufliche Identität des Autors hat.

Wenn wir verstehen wollen, wie Software in konkreten Situationen verstanden und verwendet wird, muss dieses Hineinreichen von Technik in weiter gefasste, nicht mehr technische persönliche und gesellschaftliche Bereiche berücksichtigt werden. Die Beobachtung, das heißt Aufzeichnung, Analyse und Interpretation von Nutzungssituationen, kann hierfür sensibilisieren.

Die Abb. 6.1 (nach Donick 2016, S. 313f.) zeigt die Problematik von Medium und Schnittstelle noch einmal im Zusammenhang. Vor dem Hintergrund der Nutzungssituation werden Ziele und Pläne von Nutzer*innen und Entwickler*innen als relevant gesetzt und/oder als relevant erkannt. Davor kann sich der Charakter der Software als Medium und Schnittstelle mehr oder weniger gut entfalten. Dies geschieht im stetigen Zusammenspiel von Nutzer*innen und Software, bei dem Handlungs- und Kommunikationsmöglichkeiten geprüft, gewählt und ausgeschlossen werden (Selektion). Dabei kommt es auch zur Ausdifferenzierung des Nutzer*in-Systems vor dieser Umwelt (in Luhmanns Systemtheorie spricht man Evolution des Systems). Wo Evolution nicht möglich ist (zum Beispiel, wenn Sie zu wenig Zeit haben, um eine für Sie momentan zu komplexe Umwelt zu bearbeiten), wird mitunter versucht, durch Zwang (vgl. Luhmann 2009, S. 75) zu einer Lösung zu gelangen, etwa, wenn Nutzer*innen eine nicht mehr reagierende Software durch hektische und wiederholte Mausklicks zum Funktionieren bewegen wollen. Obwohl dies im Einzelfall erfolgreich sein mag, ist der Medien- und Schnittstellencharakter der Software in der Situation doch zweifelhaft. Ist dies wiederholt zu beobachten, steht auch die Qualität der Software selbst in Frage. Dazu kommen wir nun.

6.3 Software-Qualität und Normen

6.3.1 Quality of Interaction

„Qualität ist“, so die übliche Definition nach der ISO-Norm 8402, „die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht.“ (Zuser et al. 2001, S. 284). Diese Eigenschaften und Merkmale sollten messbar und/oder beobachtbar sein (ITU 2008, S. 3). Wenn Sie in dieem Sinne trotz des qualitativen Charakters der Verfahren in diesem Buch Ihre Analysen von Nutzungssituationen auf wenige Zahlen herunterbrechen müssen, dann ist möglicherweise ein Verfahren geeignet, das als Quality of Interaction (QoI) bezeichnet wird (vgl. Donick et al. 2012; Daher et al. 2010).

QoI wurde in Ergänzung zu Quality of Service (QoS, Dienstgüte) und Quality of Experience (QoE, Erfahrungsgüte, vgl. Killki 2008) entwickelt, um den Erfolg menschlicher Kommunikation und Kooperation in CSCW (Computer Supported Cooperative Work)-Situationen zu bestimmen. Während QoS technische Parameter der Situation im Blick hat (im engeren Sinne vor allem Parameter beteiligter technischer Kommunikationsnetze, im weiteren Sinne auch messbare Eigenschaften der Situation insgesamt wie Dauer, Nach-

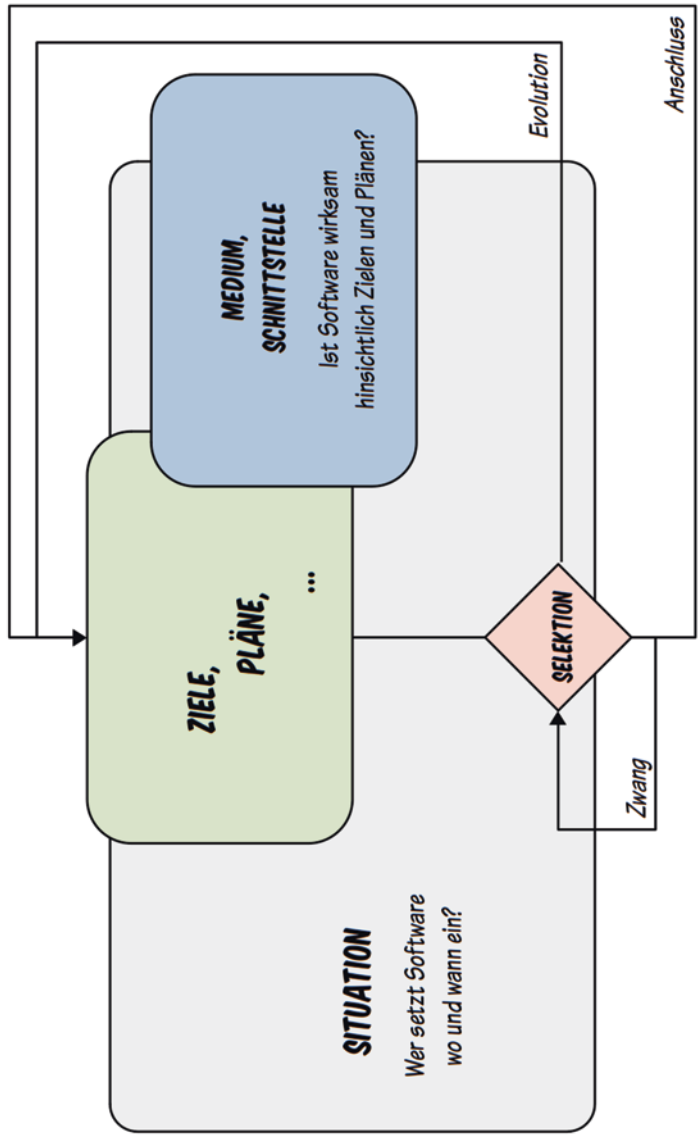

Abb. 6.1 Medium und Schnittstelle in der Schleife der Technikutzung

richtenanzahl, Datenmenge usw., vgl. ITU 2001 und ITU 2008) und QoE die subjektive Sicht von Kommunikationsteilnehmern erfasst, gibt QoI den Blick von außen auf die Situation wieder.

In Donick et al. (2012) definieren wir QoI wie folgt: „QoI bezieht Bewertungen von QoS auf Interaktionen, die auf Anwendungsebene zwischen technischen und menschlichen Kommunikationspartnern stattfinden“ (Donick et al. 2012, S. 109). „Die Messung soll die Frage beantworten, wie effizient und erfolgreich die zur Verfügung stehenden Ressourcen auf Anwendungsebene genutzt werden, z. B. von menschlichen Nutzern zur Lösung ihrer jeweiligen Aufgaben“ (ebd., S. 108). Ein besonderer Fokus liegt auf einem „zu große[n] […] Zeit- und Ressourcenverbrauch bei zu geringem Erfolg“ (ebd.)

Im Folgenden modifiziere ich den ursprünglichen Vorschlag, um QoI für die Beurteilung von Situationen der Softwarenutzung fruchtbar zu machen. Für die Berechnung von QoI werden die Ergebnisse der strukturellen Analyse numerisch aufbereitet. Von den vorgestellten Analyseverfahren eignet sich insbesondere die Kohärenzanalyse (vgl. Abschn. 5.3.2) für die Berechnung von QoI, weil in ihr schon eine numerische Bewertung des Zusammenhangs kommunikativer Spuren der Situation vorgenommen wird.

Eine praktische Anwendungsmöglichkeit von QoI in diesem Sinne ist etwa die Ermittlung einer Vergleichsbasis, anhand derer unterschiedliche Variationen von Nutzungssituationen gemessen werden können. Sie können beispielsweise die QoI bei Nutzung einer Softwareversion ermitteln und dann mit der QoI vergleichen, die mit anderen Versionen der Software oder Konkurrenzprodukten ermittelt wird. Sie können auch dieselbe Softwareversion mit unterschiedlichen Proband*innen-Gruppen testen. Dies geht schon fast in die Richtung hypothesengeleiteter quantitativer Forschung. Um den Vorteil von QoI allerdings auszuschöpfen, muss Ihre Proband*innenzahl wesentlich größer sein als bei qualitativen Analysen üblich – wenn die beobachteten Situationen sehr lange dauern, kann der Bewertungsaufwand entsprechend hoch sein.

Für die Erklärung der QoI-Ermittlung sind zunächst einige Definitionen nötig:

- *Session (Sitzung):* „Ein aufgabenorientierter Kommunikationsprozess“ (ebd., S. 108) zwischen Sender und Empfänger, zum Beispiel Nutzer*in und Software. Die Analyse der Globalstruktur (vgl. Abschn. 5.2) nimmt die Session in den Blick.
- *Subsession (Teilsitzung)*: „eine logische Gruppierung von Transaktionen innerhalb einer Session“ (ebd.), zum Beispiel ein Teil der Nutzungssituation, in der ein*e Nutzer*in gerade mit Hilfe der Software eine bestimmte abgegrenzte Aufgabe erfüllen will. Die einzelnen Abschnitte, die Sie bei der Analyse der Globalstruktur ermittelt haben, sind Beispiele für Subsessions.
- *Transaktion (Paarsequenz)*: „die basale Interaktionseinheit in einem Kommunikationsprozess“ (ebd.), mindestens aus Anfrage (request) und Antwort (response) bestehend, zum Beispiel ein von der Software eingeblendetes Dialogfenster, das ein*e Nutzer*in durch Auswahl von „OK“ oder „Abbrechen“ bearbeiten muss. Paarsequenzen untersuchen Sie bei der Analyse der Lokalstruktur (vgl. Abschn. 5.3).
- *Nachricht (Mitteilung):* „die grundlegende Einheit für den Datenaustausch“, wie die einzelne Anfrage oder Antwort. Sie ist ebenfalls Teil der Analyse der Lokalstruktur.

Da Donick et al. (2012) sowohl technische Kommunikationsnetze als auch menschliche Kommunikation im Blick haben, unterscheiden sie low-level QoI (die im Wesentlichen QoS-Parameter aufgreift) von high-level QoI (die bestimmte Parameter der low-level QoI für die Bewertung des Anwendungserfolgs nutzt; ebd., S. 109). Für unsere Zwecke verzichte ich auf diese Differenzierung. Ich spreche weiter allgemein von QoI. Zur Ermittlung der QoI einer Nutzungssituation benötigen wir wenigstens folgende Parameter (vgl. für alle ebd.):

- *Session Duration (SD, Sitzungsdauer)*: die zeitliche Dauer der beobachteten Nutzungssituation; auch als Subsession Duration/Teilsitzungsdauer möglich, wenn in der Analyse nur eine Teilsitzung betrachtet wird
- *Session Data Exchange Rate (SDXR, Datentransferrate)*: die Datenmenge je Nachricht oder Transaktion der betrachteten Sitzung oder Teilsitzung
- *Session Coherence (SC, Sitzungskohärenz)*: der in der Kohärenzanalyse ermittelte semantische Zusammenhang (Kohärenz) der beobachteten Nutzungssituation; auch als Subsession Coherence/Teilsitzungskohärenz möglich, wenn in der Analyse nur eine Teilsitzung betrachtet wird
- *Session Actors (SA, Sitzungsteilnehmer)*: die Anzahl der an der Situation beteiligten Akteure. Im Fall menschlicher Nutzer*innen kann SA relevant sein, wenn in der beobachteten Situation nicht nur einzelne Personen auftreten, sondern etwa im Paar oder im Team kooperieren sollen. Auch die klar abgrenzbare Nutzung von mehr als einer Software in der Situation durch eine oder mehrere Personen kann die Einbeziehung von SA rechtfertigen.

Der Grundansatz hinsichtlich QoI ist nun, dass eine hohe Kohärenz bei gleichzeitig geringer Sitzungsdauer und geringer Datenmenge eine hohe QoI zur Folge hat. Die von Donick et al. (2012, S. 109) vorgeschlagene Formel zur Berechnung der (high-level)-QoI lautet dementsprechend (Formel 6.1):

$$q_h = \frac{1}{n}\left(\frac{c}{sd}\right) \tag{6.1}$$

Formel 6.1 Berechnung von QoI. Dabei sind n = Anzahl der Akteure (SA); c = Sitzungskohärenz (SC); s = Datenmenge (SDXR); d = zeitliche Dauer (SD). (Aus: Donick et al. 2012, S. 109)

Um diese Formel anzuwenden, setzen Sie die Ergebnisse der Kohärenzanalyse als *c* ein und ergänzen die weiteren Parameter. Da je nach gewählten Einheiten das Ergebnis sehr klein werden kann, bietet es sich zur besseren Handhabung an, das Ergebnis am Ende mit 1000 zu multiplizieren.

Zur Demonstration greife ich hier noch einmal auf das Beispiel 5.6 aus dem letzten Kapitel zurück, für das die Kohärenzbewertung in Tab. 6.1 gemacht wurde:

Das Beispiel können wir als Subsession aus einer größeren Nutzungssituation ansehen. Wenn Sie echte Transkripte ‚Lauten Denkens' untersuchen, werden Sie sehr viele solcher

Tab. 6.1 Kohärenzwerte aus Beispiel 5.6

		1	**2**	**3**	**4**	**5**
		So …	Hm …	Was …	„Auf …“	Aber …
1	So, dann klicke ich mal … auf „Speichern“ (… 12s)					
2	Hm, wieso macht … der jetzt nichts mehr? (… 3s)	2				
3	Was? „Die Datei konnte nicht gespeichert werden“? (… 4s)	2	2			
4	„Auf Laufwerk C: ist nicht genug“ … „SPEICHERPLATZ vorhanden“?	1	1	2		
5	Aber der Stick hat doch genug …	1	−2	2	2	
6	Das KANN doch gar nicht SEIN!	−2	1	2	2	2

abgrenzbaren Bereiche erkennen. Je nach Bedarf können Sie den Mittelwert der Kohärenz für die einzelnen Bereiche oder für die gesamte Sitzung ermitteln. Im Beispiel ist die mittlere Kohärenz $c = 1{,}2$. Die Datenmenge der von Jack und der Software produzierten Äußerungen umfasst abzüglich Leerzeichen und Annotationen $s = 193$ Zeichen. Für deren Produktion veranschlagen wir inkl. der Pausen ca. $d = 38$ Sekunden. Wenn wir diese Werte direkt einsetzen (und das Ergebnis zur besseren Handhabung am Ende mit 1000 multiplizieren), erhalten wir eine QoI von $q_h = 0{,}16$. „Aha“, werden Sie sich nun vielleicht fragen, „Und was können wir mit so einem Wert anfangen? Ist 0,16 nun gut oder schlecht?“

Tatsächlich bringt uns der einzelne Wert nur etwas, wenn wir ihn mit einer Bezugsnorm verknüpfen können. Deswegen können auch die Einheiten der in die Formel eingesetzten Werte unterschiedlich sein, solange Sie für alle Proband*innen, für die Sie QoI ermitteln, dieselben Einheiten verwenden und bei der Dokumentation von QoI deutlich machen. Mögliche Bezugsnormen zur Einordnung von QoI können sein:

- *Soziale Bezugsnorm*: Hier setzen Sie die QoI-Werte einer Analyse in Bezug zu den QoI-Werten anderer Analysen. Stellen Sie sich zum Beispiel vor, dass neben Jack auch seine Freundin Jill dieselbe Situation bearbeitet hat. Auch Jill wollte ein Dokument speichern. Im Gegensatz zu Jack war Jill aber in der Lage, den richtigen Speicherort auszuwählen, sodass keine Fehlermeldung ausgegeben wurde. Die produzierte Datenmenge und die nötige zeitliche Dauer wären also geringer gewesen, während die Kohärenz der Situation höher gewesen wäre. Sei in Jills Fall die Datenmenge nur $s = 40$ Zeichen, die Dauer nur $d = 10$ Sekunden und die mittlere Kohärenz $c = 1{,}8$. Dann ist die QoI für Jill $q_h = 4{,}5$, also wesentlich höher als bei Jack. Nehmen wir Jacks Ergebnis von 0,16 als Minimum eines größeren Samples, und Jills Ergebnis als dessen Maximum, dann haben wir eine soziale Bezugsnorm, an der sich Ergebnisse weiterer Nutzer*innen messen lassen. So lässt sich dann zum Beispiel ermitteln, ob Jack ein negativer Ausreißer (nach unten) oder Jill ein positiver Ausreißer (nach oben) ist, wo also die für das Gesamtsample übliche Norm liegt.
- *Technische Bezugsnorm*: Hier setzen Sie die QoI-Werte der Analyse einer Software in Bezug zu den QoI-Werten der Analyse einer anderen Software. (Statt zweier komplett unterschiedlicher Programme können Sie auch unterschiedliche Entwürfe einer Benut-

zerschnittstelle vorlegen, um etwa den ‚besseren' zu identifizieren). Stellen Sie sich zum Beispiel vor, dass Jack und Jill und ihre Freunde zunächst mit Textverarbeitung A ihre Aufgaben erfüllen. Sie ermitteln für q_h ein Minimum von 0,3 und ein Maximum von 3,7. Nun setzen Sie Ihre Proband*innen an eine andere Textverarbeitung B und lassen dort dieselben Aufgaben bearbeiten. Nun ermitteln Sie für q_h ein Minimum von 1,2 und ein Maximum von 3,5. Das heißt, dass sich die maximal erreichbare QoI dieser Nutzer*innengruppe bei beiden Programmen nicht wesentlich unterscheidet, dass aber die starken Ausreißer nach unten verschwunden sind – dies kann ein Zeichen dafür sein, dass die Bearbeitung mit Textverarbeitung B für die beobachteten Nutzer*innen insgesamt erfolgreicher war.

- *Ergebnisqualität als Bezugsnorm*: Sie können QoI auch in Bezug zu einer – anderweitig zu ermittelnden – Qualität des Arbeitsergebnisses setzen, das mit einer Software erzielt wurde. Diese Sicht geht davon aus, dass die Form und Funktion eines Werkzeugs einen Einfluss auf das mit Hilfe des Werkzeugs erzielte Ergebnis hat. Idealerweise geht eine hohe QoI dann mit einer hohen Ergebnisqualität einher. Das Gegenteil, also eine niedrige QoI bei trotzdem hoher Ergebnisqualität, „ist möglich, wenn die […] zu bearbeitende Aufgabe zwar erfüllt wurde, die dafür nötigen Interaktionen jedoch zu lange gedauert und zu viele Ressourcen genutzt haben." (ebd.,S. 110). Der dritte Fall, eine niedrige Ergebnisqualität bei hoher QoI, kann eintreten, wenn die verwendete Software zwar Medium (im Sinne von Abschn. 6.2.1) war, also durchgängig zu Unterscheidungen motiviert hat (d. h. hohe Kohärenz und durchgängige strukturelle Kopplungen waren zu beobachten), die Software aber dennoch für die Arbeitsaufgabe keine geeignete Schnittstelle (im Sinne von Abschn. 6.2.2) war.

Zusammenfassend ist QoI ein Werkzeug, mit dem Sie die Ergebnisse Ihrer Analysen auf einen Punkt bringen können. Allerdings sollten Sie vermeiden, QoI als Endziel Ihrer Untersuchungen anzusehen – aus dem Kontext der Analyse gerissen, besteht die Gefahr, dass man QoI zu schnell als absolutes Qualitätsurteil missversteht, und gerade das ist QoI, wie gezeigt, nicht. QoI ist stets nur in Hinblick auf Bezugsnormen zu verstehen – auf andere Menschen, auf technische Alternativen, auf Arbeitsergebnisse. Beachten Sie diesen Hinweis, kann QoI eine hilfreiche Ergänzung zu den ansonsten qualitativen Untersuchungen sein.

6.3.2 Gestaltungsnormen

Einen weiteren praxisnahen Bezugspunkt der Beobachtung von Nutzer*innen finden wir in der Norm „Grundsätze der Dialoggestaltung" (DIN EN ISO 9241-110). Damit knüpfen wir unsere Untersuchungen an den Bereich der Usability-Forschung an. Die ISO 9241-110 nennt sieben Grundsätze, die – wenn sie sich denn aus Nutzer*innen-Sicht als erfüllt darstellen – schon viel zum Ideal transparenter Software (vgl. Kap. 1) beitragen können. Diese Grundsätze sind:

- *Aufgabenangemessenheit*: Die Funktionalität der Software soll Nutzer*innen unterstützen, die intendierten Nutzungsziele mit möglichst wenig Interaktionsaufwand zu erreichen. Sie müssen sich zum Beispiel nicht erst umständlich durch Menüs und Untermenüs klicken, um die gewünschte Funktion zu finden.
- *Selbstbeschreibungsfähigkeit*: Die Software soll Nutzer*innen deutlich machen, wie sie zu bedienen ist und was sie tut, und sie soll das durch geeignete Hilfetexte und Rückmeldungen unterstützen. Wenn Sie zum Beispiel die Dialogfelder und Nachrichten der Software lesen, wissen Sie damit etwas anzufangen.
- *Lernförderlichkeit*: Die Software soll das möglichst schnelle Erlernen ihrer Bedienung und Funktionalität fördern. Zum Beispiel leiten Assistenten Sie schrittweise an oder bei Bedarf stehen Tutorials zur Verfügung.
- *Steuerbarkeit*: Die Steuerung der Software geschieht durch die Nutzer*innen; sie haben die Kontrolle über die Software. Zum Beispiel sehen Sie zu jeder Zeit, wie weit die Software mit der Bearbeitung ist und Sie haben die Möglichkeit, Vorgänge abzubrechen, oder zu pausieren und später fortzufahren.
- *Erwartungskonformität*: Die Bedienung der Software soll konsistent sein und den Erwartungen entsprechen, die Nutzer*innen an sie haben. Zum Beispiel entspricht der Aufbau von Menüs, Dialogen und Symbolleisten gewissen Standards, denen Sie schon bei anderer Software begegnet sind.
- *Individualisierbarkeit*: Die Bedienung der Software soll sich an die individuellen Bedürfnisse der Nutzer*innen anpassen lassen. Zum Beispiel können Sie Symbolleisten so anordnen, wie Sie am besten damit zurechtkommen.
- *Fehlertoleranz*: Die Software darf durch Fehler nicht instabil werden, muss sich weiter bedienen lassen und Nutzer*innen die leichte Korrektur von Fehlern erlauben. Zum Beispiel lässt eine Fehlermeldung Sie nicht ratlos zurück, sondern erklärt in für Sie verständlichen Worten die Ursache des Fehlers und was Sie konkret dagegen tun können.

Wenn diese Anforderungen weitestgehend erfüllt sind, kann man davon ausgehen, dass die betreffende Software ein hohes Potenzial bietet, für ihre jeweils angedachten Nutzungssituationen geeignet zu sein. Doch obwohl eine Norm vorschreibenden Charakter hat und sie davon ausgeht, dass man ihr gemäß handeln oder, wie im Fall der ISO 9241-110, Technik in ihrem Sinne gestalten kann, so ist die tatsächliche Geltung der Norm doch eine Beobachtung der betroffenen Menschen. Als Entwickler*in können (und sollten) Sie versuchen, Ihr Produkt entsprechend der genannten Grundsätze zu gestalten, aber das garantiert nicht, dass sich die gewählte Gestaltung für alle denkbaren Nutzer*innen und Nutzungssituationen eignet.

Die Beobachtung von Nutzer*innen und die Analyse der Beobachtungen kann Ihnen dabei helfen, potenzielle Problemstellen auszumachen, die Sie sonst vielleicht übersehen hätten. Aber wie kommen wir aus den Ergebnissen der strukturellen Analyse zu Aussagen über die sieben genannten Anforderungen der ISO 9241-110? Glücklicherweise haben Sie schon alles gelernt, was Sie dazu brauchen.

Tab. 6.2 Bezugspunkte zur DIN EN ISO 9241-110

Anforderung	Bezugspunkt	Bedeutung
Aufgabenangemessenheit	Kopplung, Schnittstelle	Der Erhalt des Nutzer*in-Systems kann nur dann erfolgen, wenn die Software als angemessen für die Ziele des Systems erscheint.
Selbstbeschreibungsfähigkeit	Medium	Die Eigenschaft von Software, sich und ihre Funktionen selbst zu beschreiben, beeinflusst in großem Maße ihre Selektivität, d. h. ihre Eigenschaft, zu Selektionen zu motivieren.
Steuerbarkeit	Medium, Kopplung	Die Steuerung von Software geschieht durch Selektionen, die vorhandene Selektivität einschränken, wodurch letztlich der Erhalt des Nutzer*in-Systems möglich wird.
Erwartungskonformität	Kopplung, Evolution	Der Erhalt des Nutzer*in-Systems wird wesentlich durch Erwartungen und deren Bestätigung oder Enttäuschung beeinflusst. Enttäuschte Erwartungen können Evolution des Systems anregen.
Fehlertoleranz	Kopplung	Fehler in der Software können zum Bruch der strukturellen Kopplung mit dem Nutzer*in-System führen.
Individualisierbarkeit	Kopplung, Evolution	Die Möglichkeit, Software an die Bedürfnisse des Nutzer*in-Systems anzupassen, kann das Komplexitätsgefälle zwischen System und Umwelt verringern und somit Systemerhalt und Evolution erleichtern.
Lernförderlichkeit	Evolution, Kopplung	Wenn Software leicht erlernbar ist, wird Evolution des Nutzer*in-Systems erleichtert.

Eine Möglichkeit ist nämlich, Ihre Nutzer*innen selbst darüber sprechen zu lassen. Oft hinterlassen Nutzer*innen beim ‚Lauten Denken' selbst schon Hinweise darauf, in welchen Anforderungsbereichen sie akut Schwierigkeiten haben. Ihre Nutzer*innen werden vermutlich nicht von „Aufgabenangemessenheit" sprechen, aber wenn jemand fast fatalistisch feststellt: „Leider ist nichts möglich" (wie ein Proband in Donick 2016, S. 189), spricht das für sich. Sie werden auch kaum das Wort „Selbstbeschreibungsfähigkeit" vernehmen – wenn aber jemand zum Beispiel sagt: ‚Ich verstehe nicht, was der von mir will', dann haben Sie einen deutlichen Hinweis. Ähnlich können Sie auch für die anderen Bereiche vorgehen. Systematisch wird dies, wenn Sie diese Normbezüge bei der thematischen Analyse der Lokalstruktur ermitteln (vgl. Abschn. 5.3.1).

Ein anderer Ansatz ist es, die sieben Grundsätze der Dialoggestaltung auf die in diesem Buch immer wieder verfolgte systemtheoretische Perspektive abzubilden, d. h. strukturelle Kopplung, Evolution (des Nutzer*in-Systems, vgl. Kap. 3), Medium und Schnittstelle. In Tab. 6.2 sind die Anforderungen der Norm passenden systemtheoretischen Konzepten zugeordnet.

Wenn Sie strukturelle Kopplungen nachzeichnen, den Mediencharakter der Software in der Nutzungssituation herausarbeiten und die Eignung der Software als Schnittstelle beurteilen, erhalten Sie Anhaltspunkte, um die Erfüllung oder Nicht-Erfüllung der Norm einzuschätzen bzw. abzuleiten, was für die Erfüllung der Norm notwendig wäre. Die Tabelle zeigt, worauf Sie sich bei den einzelnen Anforderungen konzentrieren sollten:

- Strukturelle Kopplungen als solche geben zunächst Hinweise auf die Steuerbarkeit und Fehlertoleranz der Software – wenn zum Beispiel ein Fehler die Nutzung nur kurz unterbricht und sie danach schnell weitergeht.
- Hinsichtlich der Steuerbarkeit beurteilen Sie die Medialität der Software (vgl. Abschn. 6.2.1). Dies gilt auch für die Selbstbeschreibungsfähigkeit.
- Wollen Sie wissen, ob die Software aufgabenangemessen ist, sollten Sie untersuchen, ob die analysierten Kopplungsprozesse auf einen Schnittstellencharakter der Software hindeuten (vgl. Abschn. 6.2.2).
- Erwartungskonformität, Individualisierbarkeit und Lernförderlichkeit erkennen Sie an Kopplungen, die auf Evolution des Nutzer*in-Systems hindeuten (also Weiterentwicklung der eigenen Erwartungen, sowie Anpassung der Software an die Erwartungen).

6.4 *Nachbemerkung: Muss ich wirklich* das alles *machen?*

Es wurde in diesem Buch schon mehrfach erwähnt, dass es auf Ihre Fragestellung ankommt, welche der hier vorgeschlagenen Analyseverfahren und Interpretationsansätze für Sie geeignet sind. Sie haben einen Werkzeugkasten erhalten, der Ihnen hilft, unterschiedliche Zusammenhänge der situationsgebundenen Softwarenutzung verstehen. In dem Werkzeugkasten befinden sich drei Ansätze für die strukturelle Analyse (Themen, Kohärenzbeziehungen, Kopplungen) und vier Ansätze für die Interpretation (Medium, Schnittstelle, QoI, Gestaltungsnormen). Ich habe Ihnen gezeigt, welche Analyseansätze ich für welche Interpretationsansätze für sinnvoll halte, aber prinzipiell sind Sie frei darin, alles so zu kombinieren, wie es sich für *Sie* als sinnvoll zeigt. Denn wie jedes Werkzeug, erweisen sich auch die vorgestellten Ansätze erst beim Einsatz als nützlich oder nicht nützlich.

Und denken Sie daran: Manchmal liegt die Leistung eines Werkzeugs auch in seiner Zweckentfremdung. Da die grundlegenden Eigenschaften menschlicher Kommunikation stets in ähnlicher Weise auftreten, können Sie die Analyse- und Interpretationsansätze flexibel einsetzen. Wir haben uns in diesem Buch immer Nutzungssituationen angeschaut – aber Sie können mit denselben Ansätzen auch ganz andere Kommunikationssituationen untersuchen, etwa die Zusammenarbeit im Team, die Einarbeitung in fremden Programmcode, die Eignung von Dokumentationen für ein Projekt und vieles mehr.

Wenn Sie so strukturiert vorgehen, wie ich es Ihnen hoffentlich vermitteln konnte, werden Sie mit Hilfe der gezeigten Ansätze in allen diesen Situationen interessante Erkenntnisse gewinnen. Probieren Sie es aus!

Literatur

Baecker, Dirk: Studien zur nächsten Gesellschaft. Frankfurt/Main 2007.

Brinker, Klaus / Sager, Sven F.: Linguistische Gesprächsanalyse. Eine Einführung. Berlin 2006.

Daher, Robil/Donick, Mario/Krohn, Martin/Tavangarian, Djamshid: Quality of Interaction: an Alternative Model for QoS in Interplanetary and Deep-Space Communication Networks. In: Proceedings of 28th AIAA International Communications Satellite Systems Conference (ICSSC-2010), 30.08–02.09.2010, Anaheim/USA, 2010.

Donick, Mario: „Offensichtlich weigert sich Facebook, mir darauf eine Antwort zu geben“: Strukturelle Analysen und sinnfunktionale Interpretationen zu Unsicherheit und Ordnung der Computernutzung. Hamburg 2016.

Donick, Mario: Die Unschuld der Maschinen. Technikvertrauen in einer smarten Welt. Wiesbaden 2019.

Donick, Mario/Daher, Robil/Schwelgengräber, Wiebke/Krohn, Martin/Tavangarian, Djamshid: Messung und Oaptimierung der Interaktionsgüte (Quality of Interaction) zeitverzögerter CSCW-Anwendungen. In: Praxis der Informationsverarbeitung und Kommunikation, Band 35, Heft 2, 2012, S. 107–112.

DIN EN ISO 9241-110 Grundsätze der Dialoggestaltung; URL: https://www.iso.org/standard/38009.html (letzter Abruf: 10.09.2019).

ITU 2008: International Telecommunications Union: Recommendation E.800 : Definitions of terms related to quality of service; URL: https://www.itu.int/rec/T-REC-E.800-200809-I/en (letzter Abruf: 10.09.2019).

ITU 2001: International Telecommunications Union: Recommendation G.1000: Communications Quality of Service: A framework and definitions; URL: https://www.itu.int/rec/T-REC-G.1000-200111-I/en (letzter Abruf: 10.09.2019).

Killki, K.: Quality of Experience in Communication Ecosystem In: Journal of Universal Computer Science, vol. 14, no. 5, 2008, S. 615–624.

Krotz, Friedrich: Mediatisierung. Fallstudien zum Wandel von Kommunikation. Wiesbaden 2007.

Luhmann, Niklas: Die Unwahrscheinlichkeit der Kommunikation. In: Soziologische Aufklärung 3, 1981, S. 25–34.

Luhmann, Niklas: Die Wissenschaft der Gesellschaft. Frankfurt/Main 1992.

Luhmann, Niklas: Vertrauen. Stuttgart 2009.

Zuser, Wolfgang/Biffl, Stefan/Grechenig, Thomas/Köhle, Monika: Software Engineering mit UML und dem Unified Process. München 2001.

Empfehlungen zum Weiterlesen

Aus der in diesem Buch zitierten Literatur sind folgende Werke besonders hervorzuheben. Sie können parallel oder ergänzend zu vorliegendem Buch gelesen werden:

Lucy Suchman: Human-Machine Reconfigurations. Plans and Situated Actions 2nd Edition. Cambridge 2007. Die Autorin war eine der ersten, die tatsächliches menschliches Handeln im Umgang mit technischen Geräten beobachtet hat, um herauszufinden, wie Menschen sich gemeinsam ihre durch Technik geprägte Wirklichkeit konstruieren. Suchman zeigt an ihren Beobachtungen, dass Pläne nur situationsspezifisch wirksam werden können, dass Pläne deshalb oft unvollständig sind und dass sie bei der Umsetzung modifiziert werden müssen.

Heinz von Foerster: Wissen und Gewissen. Frankfurt/Main 1993. In zahlreichen gut lesbaren und teils leicht ironischen Aufsätzen zeigt der Ingenieurwissenschaftler und Konstruktivist, dass Steuerung und Kontrolle ein Problem doppelter Beobachtung ist (= Beobachter beobachten Beobachter; Kybernetik 2. Ordnung). Dieses Denken war unter anderem für die soziologische Systemtheorie (Niklas Luhmann, Dirk Baecker) von Bedeutung.

Niklas Luhmann: Vertrauen. Stuttgart 2009. Der Soziologe erläutert, welche Funktion Vertrauen für Menschen besitzt, wie Menschen versuchen, Vertrauen herzustellen und was sie tun, wenn Vertrauen nicht möglich ist. Der kleine Band ist auch ohne tiefere Kenntnisse von Luhmanns Systemtheorie gut verständlich.

Dan Sperber und Deidre Wilson: Relevance. Communication and Cognition. Oxford 1995. Die Autoren zeigen aus sprachphilosophischer Sicht, warum die Einstufung von Reizen und Annahmen als mehr oder weniger relevant entscheidend für das Verständnis menschlicher Kommunikation ist, und dass die Relevanz eines Reizes oder einer Annahme mit dem Aufwand ihrer Verarbeitung zusammenhängt.

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8

Stichwortverzeichnis

M. Donick, *Nutzerverhalten verstehen – Softwarenutzen optimieren*,
https://doi.org/10.1007/978-3-658-28963-8