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