
Nutzerverhalten verstehen – Softwarenutzen optimieren

Mario Donick

Nutzerverhalten verstehen – Softwarenutzen optimieren

Kommunikationsanalyse bei der
Softwareentwicklung

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 Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2020

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von allgemein beschreibenden Bezeichnungen, Marken, Unternehmensnamen etc. in diesem Werk bedeutet nicht, dass diese frei durch jedermann benutzt werden dürfen. Die Berechtigung zur Benutzung unterliegt, auch ohne gesonderten Hinweis hierzu, den Regeln des Markenrechts. Die Rechte des jeweiligen Zeicheninhabers sind zu beachten.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag, noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

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 kommunizieren“, 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 kommunizieren. 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
30. September 2019

Mario Donick

Literatur

Rosa, Hartmut: Unverfügbarkeit. Wien/Salzburg 2019.

Inhaltsverzeichnis

Teil I Grundlagen – Warum es wichtig ist, Softwarenutzung zu beobachten

1	Einleitung: Was Software-Qualität mit menschlicher Kommunikation zu tun hat	3
1.1	Software als Werkzeug	3
1.2	Warum Software oft nicht-trivial ist und warum das eine Herausforderung ist	5
1.3	Die Bedeutung der Nutzungssituation	11
1.3.1	Situation, Handlung und Verhalten	11
1.3.2	Softwarequalität als situationsabhängiger Wert	14
1.4	Menschliche Kommunikation bei Software-Entwicklung und -Nutzung	15
1.5	Transparente Software als gesellschaftliches Ideal	18
	Literatur	21
2	Kommunikation bei der Softwareentwicklung	23
2.1	Kommunikation: Vorausgesetzt, gefordert, aber selten reflektiert	23
2.1.1	Beispiel 1: <i>Wikipedia</i> -Artikel „Extreme Programming“	24
2.1.2	Beispiel 2: Ein Lehrbuch zur Agilen Softwareentwicklung	27
2.1.3	Was uns die Beispiele verdeutlichen	29
2.2	Modelle menschlicher Kommunikation	30
2.2.1	Kommunikation als Übertragung vom Sender zum Empfänger	30
2.2.2	Kommunikation als Beziehung und Haltung	32
2.2.3	Kommunikation als Differenzmanagement	38
2.3	Wozu brauchen wir das?	42
	Literatur	44
3	Über die Beziehung von Nutzer*in, Software und Nutzungssituation	47
3.1	<i>Ziel von Entwicklung</i> : Relevante Werkzeuge herstellen	47
3.1.1	Relevanz als Kommunikationsmerkmal	47
3.1.2	Software und Relevanz	49
3.2	Zweiseiten-Formen der Softwarenutzung	52
3.2.1	Reproduktion und Störung – Nutzer*innen als Systeme	52

3.2.2	Form und Funktion – Software als Design	54
3.2.3	Plan und Abweichung – Nutzungsziele in der Nutzungssituation . . .	54
3.2.4	Einfaches Beispiel aus der Praxis	56
3.3	Wozu brauchen wir das?	58
	Literatur.	59
Teil II Anwendung – Wie wir Softwarenutzung beobachten können		
4	Softwarenutzung strukturiert beobachten	63
4.1	Was heißt „Softwarenutzung beobachten“?	63
4.2	Geeignete Fragestellungen, benötigte Daten	67
4.3	Beobachtungen planen und organisieren	72
4.3.1	Die Suche nach Proband*innen	72
4.3.2	Den Beobachtungsort festlegen	74
4.3.3	Benötigte Technik	75
4.3.4	Die richtigen Aufgaben stellen.	76
4.4	Beobachtungen durchführen, Daten erheben.	77
4.4.1	Grundsätzliches am Tag der Beobachtung.	77
4.4.2	Ethische Überlegungen	78
4.4.3	Was Menschen tun, wenn sie sich beobachtet fühlen	80
	Literatur.	82
5	Strukturelle Analyse der Beobachtungsdaten	85
5.1	Transkription und Aufbereitung.	85
5.2	Die Globalstruktur ermitteln: Sich einen Überblick verschaffen.	87
5.3	Die Lokalstruktur analysieren: In die Tiefe gehen.	88
5.3.1	Themenentwicklung (Progression)	89
5.3.2	Zusammenhang (Kohärenz).	96
5.3.3	Strukturelle Kopplung (Differenz).	102
	Literatur.	106
6	Analyseergebnisse interpretieren: Software als Medium und Schnittstelle, Quality of Interaction und Gestaltungsnormen	107
6.1	Vorbemerkung: Was heißt „Interpretation“?	107
6.2	Interpretationsbeispiele	109
6.2.1	Software als Medium	109
6.2.2	Software als Schnittstelle.	114
6.3	Software-Qualität und Normen	116
6.3.1	Quality of Interaction.	116
6.3.2	Gestaltungsnormen	121
6.4	Nachbemerkung: Muss ich wirklich das alles machen?	124
	Literatur.	125
	Empfehlungen zum Weiterlesen	125
	Stichwortverzeichnis	127