Technische Universität Dresden

Fakultät Informatik

Institut für Software und Multimediatechnik

Professur für Didaktik der Informatik

Beurteilung der didaktischen Eignung ausgewählter Programmiersprachen für den Informatikunterricht in sächsischen Gymnasien

Wissenschaftliche Arbeit zur Didaktik der Informatik
für das Lehramt an Gymnasien

vorgelegt von Dirk Köhler

geboren am 09.02.1969

Erstgutachter: Herr Dr. Holger Rohland

Zweitgutachter: Herr Jens Spröte

Dresden, Oktober 2019

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Anwendungsprogrammierschnittstelle
CSS	mehrstufige Formatvorlagen
EPA	Einheitliche Prüfungsanforderungen
FIFO	zuerst herein, zuerst hinaus
GI	Gesellschaft für Informatik
GUI	grafische Benutzeroberfläche
IDE	integrierte Entwicklungsumgebung
LIFO	zuletzt herein, zuerst hinaus
SQL	strukturierte Abfragesprache
WYSIWYG	Was du siehst, ist, was du bekommst.

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Ein Blick auf Programmiersprachen-Rankings wie den TIOBE Programming Community Index[1] oder RedMonk[2] offenbart die große Vielfalt aktueller Programmiersprachen. Die Popularität einzelner Sprachen ist zwar volatil und die Rankings entsprechen nicht strengen wissenschaftlichen Kriterien, jedoch wird deutlich, vor welchen Auswahlproblemen die Lehrkräfte im Informatikunterricht an Gymnasien stehen. Diese Arbeit greift deshalb Fragen der Auswahl der Programmiersprache für den Informatikunterricht an sächsischen Gymnasien auf und entwickelt einen Kriterienkatalog zur Beurteilung von Programmiersprachen für den Unterricht.

1.2 Thematische Abgrenzung

1.2.1 Abgrenzung hinsichtlich der Programmiermethode

Neben den textbasierten Programmierung existiert eine weitere Methode der Programmierung, die visuellen Programmierung. Mit der visuellen Programmierung wird ein Programm durch die Anordnung grafische Elemente erschaffen. Mit Blick auf den Untersuchungsgegenstand soll eine Eingrenzung auf die textbasierte Programmiersprachen vorgenommen und begründet werden. Hierzu werden zunächst die Vor- und Nachteile der textbasierten und der visuellen Programmierung dargelegt.

Förster schreibt zur visuellen Programmierung:

> *„Unter der Flagge "Low Code" segeln nun Werkzeuge, die ein Aspekt eint: Sie sollen das händische Kodieren durch visuelle Editoren, Point-&-Click-Baukästen sowie diverse Automatisierungsfunktionen ersetzen und damit die Einstiegshürde für Programmierer deutlich senken.“*[3]

Auch in der Lehre und der Schulpraxis steht die visuelle Programmierung in visuellen Entwicklungsumgebungen neben der klassischen textbasierten Programmierung und gewinnt breiten Raum. Eine weit verbreitete visuelle Programmiersprache ist *Scratch*[4].

1 Siehe https://www.tiobe.com/tiobe-index/.
2 Siehe https://redmonk.com/sogrady/2019/03/20/language-rankings-1-19/.
3 Förster, 2019.
4 Siehe https://scratch.mit.edu/.

1.2.1.1 Vorteile

Modrow befürwortet den Einstieg in den Unterricht mit visueller Programmierung und argumentiert, dass mit visueller Programmierung eine Konzentration auf inhaltliche Aspekte des Programmierens vollzogen wird.[5] Er wirft die Frage auf, „ob sie nicht in absehbarer Zeit die klassischen textbasierten Sprachen ablösen können.“[6]

Für den Programmieranfänger und die Programmieranfängerin ist vordergründig vorteilhaft, dass er bei der Nutzung einer visuellen Programmiersprache den Code nicht mit der Tastatur eingeben muss. Vielmehr stehen hierfür grafische Elemente zur Verfügung, die im Wesentlichen angeordnet werden. Syntaxfehler werden damit vermieden und die Schülerinnen und Schüler können sich auf die Logik der Programmierung konzentrieren.

Mit dieser Programmiermethode verbunden sind meist auch schnelle Erfolgserlebnisse der Schülerinnen und Schüler, denn der Ärger über Syntaxfehler – wie ein vergessenes Komma oder eine nicht geschlossene Klammer – entfällt.

1.2.1.2 Nachteile

Die Gegenposition zur visuellen Programmierung führt ebenfalls eine Vielzahl von Argumenten an. Ein sehr anschauliches Argument ist das sogenannte Deutsch Limit des US-amerikanischen Informatikers L. Peter Deutsch, der Entwickler des freien PostScript-Interpreters Ghostscript. Deutsch schrieb:

> *„Well, this is all fine and well, but the problem with visual programming languages is that you can't have more than 50 visual primitives on the screen at the same time. How are you going to write an operating system?“*[7]

Es liegt auf der Hand, dass der begrenzte Platz auf einem Bildschirm die visuelle Programmierung erschwert. Mit der visuellen Programmierung werden zwar schnell einfache Ergebnisse erreicht, aber aufwändigere Projekte können oft nur beschränkt verwirklicht werden.[8]

5 Vgl. Modrow, Seite 27.
6 Ebenda.
7 Deutsch. [„Nun, das ist alles in Ordnung, aber das Problem bei visuellen Programmiersprachen ist, dass Sie nicht mehr als 50 visuelle Grundelemente gleichzeitig auf dem Bildschirm haben können. Wie schreibst du ein Betriebssystem?“].
8 Vgl. Kohls, 2014, Seite 12.

Gleichfalls problematisch ist die Mehrdeutigkeit der grafischen Notation der algorithmischen Grundstrukturen bei der visuellen Programmierung. Wie beispielsweise in der folgenden Abbildung des Online-Editors[9] der Sprache *Scratch* ersichtlich, wird eine Zählschleife und eine Selektion mit gleichen Symbolen dargestellt.

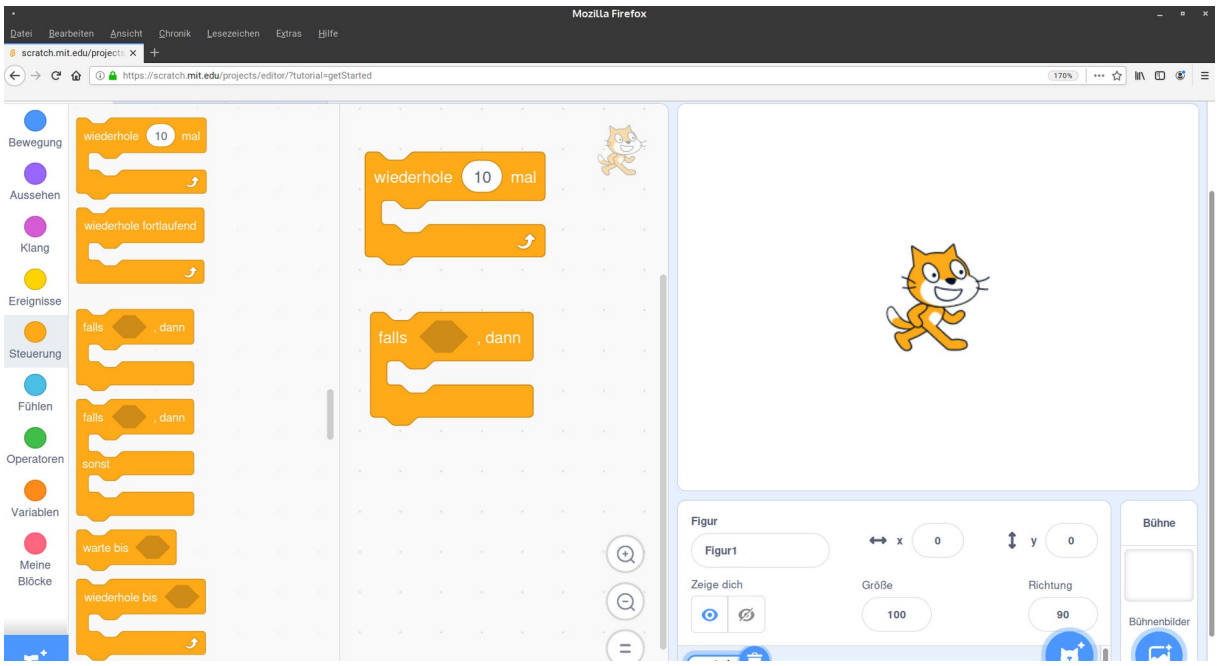

Abbildung 1: Online-Editor Scratch

Ein grundsätzliches Missverständnis dieser algorithmischen Grundstrukturen liegt deshalb nahe. Noch dramatischer wird diese Problematik bei einer mehrfachen Verschachtelung dieser algorithmischen Grundstrukturen.

Bei vielen Systemen, insbesondere den kontrollflussbasierten Systemen, existiert für jede Anweisung ein Äquivalent in der textuellen Programmierung. Syntax und Konfiguration sind in der Regel fest vorgegeben. Dadurch werden Anfänger- und Flüchtigkeitsfehler vermieden. Jedoch ist auch hier das algorithmische Denken genauso unabdingbar wie bei textuellen Sprachen.

Hinsichtlich höherer Programmierkonzepte gibt es bei der visuellen Programmierung Einschränkungen zu beachten. Meist stehen Konzepte wie Vererbung, Polymorphie oder funktionale Programmierung nicht zur Verfügung.[10]

Kohls bemängelt weiter, dass die visuelle Aufbereitung die grundsätzlichen Konzepte der Programmierung oft verschleiert und somit ein vertieftes Verständnis der Abläufe erschwert wird.[11]

9 Siehe https://scratch.mit.edu.
10 Vgl. Kohls, 2014, Seite 12.
11 Vgl. ebenda.

1.2.1.3 Fazit

Nach der Darlegung der Vor- und Nachteile ist festzustellen, dass sich visuelle Programmiersprachen als Vergleichsobjekte im Rahmen der vorliegenden Arbeit nicht eignen. Visuelle Programmiersprachen erfordern aufgrund ihrer typischen Eigenheiten andere Bewertungskriterien als textbasierte Programmiersprachen. Aus diesem Grund werden visuelle Programmiersprachen nicht mit in die Beurteilungen aufgenommen. Diese Arbeit beschränkt sich auf die Untersuchung textbasierter Programmiersprachen.

1.2.2 Abgrenzung hinsichtlich des Anwendungszwecks

Programmiersprachen können zu den verschiedenen Zwecken verwendet werden. So unterstützen bestimmte Programmiersprachen beispielsweise die Entwicklung von systemnahen Anwendungen und andere die Entwicklung von Webapplikationen.

Die vorliegende Arbeit konzentriert sich auf die Eignung zur Erreichung der im sächsische Lehrplan für Gymnasien gestellten Lernziele. Die Eignung für andere Zwecke werden nicht betrachtet.

1.3 Vorgehen und Methodik

Die Methodik der vorliegenden Arbeit basiert zunächst auf einer Analyse der didaktischen Anforderungen an eine Programmiersprache. Hierfür wird eine qualitative Auswertung der Fachliteratur vorgenommen.

Anschließend werden diese Anforderungen ergänzt durch eine Analyse von Anforderung des sächsischen, gymnasialen Lehrplans der Sekundarstufe I und II. Diese Lehrplananalyse beschränkt sich nicht nur auf die isolierte Betrachtung der einschlägigen Lernbereiche, sondern berücksichtigt gleichfalls allgemeine Grundsätze des Lehrplans und kontextuelle Zusammenhänge im Lehrplan. Damit werden sich aus dieser Analyse eventuell ergebende weitere Anforderungen mit berücksichtigt. Danach werden die Einheitlichen Prüfungsanforderungen (EPA) und die Bildungsstandards für die Sekundarstufe I und II hinsichtlich der Anforderungen für eine Programmiersprache betrachtet und ausgewertet.

Aufgrund dieser Analysen wird ein Kriterienkatalog für die Auswahl einer Programmiersprache aufgestellt. Anhand dieses Kataloges findet eine Beurteilung der ausgewählten Programmiersprachen mit Code-Beispielen statt. Diese Code-Beispiele werden aus den analysierten Anforderungen gewonnen.

5 Fazit

Die Motivation der Arbeit war es, dass eine Vielzahl von Programmiersprachen für den Informatikunterricht zur Verfügung stehen, es aber kaum Auswahlhilfen hierfür gibt. Deshalb wurde die didaktischen Eignung ausgewählter Programmiersprachen für den Informatikunterricht in sächsischen Gymnasien beurteilt.

Ausgangspunkt hierfür war die Analyse der fachdidaktischen Lehre, des sächsischen Lehrplans, der EPA und der Bildungsstandards. Mit dieser Analyse wurden Anforderungen an eine Programmiersprache für den Informatikunterricht abgeleitet. Diese Anforderung bildeten wiederum die Grundlage für einen Kriterienkatalog zur Beurteilung der Sprachen. Die Begründung des Kriterienkatalogs erfolgte mittels des bildungstheoretischen Ansatzes. Nach der Definition der Bewertungsstufen, der Gruppierung von Kriterien und der Wichtung der Kriterien stand ein Werkzeug zu Beurteilung der Programmiersprachen zur Verfügung.

Die Beurteilung der fünf Programmiersprachen zeigte im Wesentlichen die Bildung von drei Gruppen hinsichtlich der Gesamtnoten. Die Abstände zwischen den Gruppen bewegen sich in einem signifikanten Bereich.

Die erste Gruppe ist gekennzeichnet durch eine Gesamtbewertung kleiner als die Note 2,0. Diese Gruppe – in der sich die Sprache *Python* befindet – kann mit Blick auf die Bewertungsstufen aus Kapitel 3.4 mit dem Attribut „besonders schülergerecht“ versehen werden.

Die zweite Gruppe erhielt eine Gesamtbewertung nahe der Note 2,0. Diese Gruppe kann entsprechend den Bewertungsstufen aus Kapitel 3.4 als „schülergerecht“ attribuiert werden. In dieser Gruppe finden sich die Programmiersprachen *Pascal* und *Java*.

In der dritten Gruppe wiederum bewegen sich die Gesamtergebnisse nahe der Note 2,5. Entsprechend den Bewertungsstufen aus Kapitel 3.4 können die Sprachen in dieser Gruppe als „geeignet“ betrachtet werden. *Javascript* und *PHP* sind in dieser Gruppe.

Betrachtet man den zur Beurteilung entwickelte Kriterienkatalog, so ist festzuhalten, dass sich dieser als ein taugliches Beurteilungsinstrument erwiesen hat. Da dieser Kriterienkatalog auf dem Checklistenverfahren basiert, können damit später ähnlich gelagerten Situationen oder Sachverhalte abermals entschieden werden. Dieses Checklistenverfahren bietet darüber hinaus den Vorteil, dass auf diese Weise Erfahrungen aus mehreren Entscheidungsprozessen zusammengetragen werden und damit die Liste optimiert werden kann.[138]

138 Vgl. Gabler-01.

Darüber hinaus zeigte sich, dass die Wichtung der Kriterien ein Mittel ist, mit dem auf verschiedene Unterrichtsszenarien reagiert werden kann. Wertet man beispielsweise die Komplexität ab und die Alltagsorientierung auf, so verbessern sich die Ergebnisse der dritten Gruppen. Ein entsprechendes Unterrichtsszenario wäre gegeben, wenn die Komplexität einen geringeren Einfluss auf den Lernprozess hat. Dies könnte in leistungsstarken Klassen der Fall sein.

Insgesamt bieten die hier vorgenommene Bewertungen und der entwickelte Kriterienkatalog als Bewertungsinstrument dem Lehrpersonal ein taugliche Entscheidungsgrundlage für die Auswahl einer Programmiersprache. Dabei wird nicht nur die Einhaltung des Lehrplans sichergestellt, vielmehr wird auch ein flexibles Reagieren auf besondere Konstellationen und Bedürfnisse einer konkreten Lerngruppe ermöglicht.

Literaturverzeichnis

Ackermann, Philip: JavaScript : Das umfassende Handbuch für Einsteiger, Fortgeschrittene und Profis. Inkl. ECMAScript 6, Node.js, Objektorientierung und funktionaler Programmierung. 2. Auflage. Bonn: Rheinwerk, 2018.
zitiert: Ackermann, 2018

Canneyt, Michael van: Lazarus : Klassenbibliothek und IDE. 2. aktualisierte. Böblingen: C&L Computer und Literaturverlag, 2011.
zitiert: Canneyt, 2011

Claus, Volker: Duden - Informatik A - Z : Fachlexikon für Studium, Ausbildung und Beruf. 4. überarbeitete und aktualisierte. Mannheim: Dudenverlag, 2006.
zitiert: Claus, 2006

Deutsch, L. Peter: Comp.Lang.Visual - Frequently-Asked Questions List : Q12: What is the Deutsch Limit? URL: ftp://rtfm.mit.edu/pub/usenet/news.answers/visual-lang/faq.
zuletzt geprüft: 19.08.2019
zitiert: Deutsch

Ehses, Erich: Paradigmen der Programmierung (Teil 1) : Funktionale Programmierung und Logikprogrammierung. FH Köln, Campus Gummersbach, Wintersemester 2014/2015,
URL: http://www.gm.fh-koeln.de/ehses/paradigmen/pp1.pdf.
zuletzt geprüft: 19.07.2019
zitiert: Ehses, 2014

Förster, Moritz: Low Code: Brauchen wir bald keine Programmierer mehr?. iX-Magazin,
URL: https://www.heise.de/ix/meldung/Low-Code-Brauchen-wir-bald-keine-Programmierer-mehr-4481355.html.
zuletzt geprüft: 19.08.2019
zitiert: Förster, 2019

Fuchs, Paul: JavaScript Programmieren für Einsteiger : Der leichte Weg zum JavaScript-Experten! 1. Auflage. Landshut: BMU Media, 2019.
zitiert: Fuchs, 2019

Gabler Wirtschaftslexikon : Checklistenverfahren.
URL: https://wirtschaftslexikon.gabler.de/definition/checklistenverfahren-27389/version-251045.
zuletzt geprüft: 19.07.2019
zitiert: Gabler-01

Gesellschaft für Informatik (GI) e. V. (Herausgeber): Bildungsstandards Informatik für die Sekundarstufe II. In : LOG IN, Beilage, 36. Jahrgang (2016), Heft Nr. 183/184.
zitiert: GI, 2016

Gesellschaft für Informatik (GI) e. V. (Herausgeber): Grundsätze und Standards für die Informatik in der Schule : Bildungsstandards Informatik für die Sekundarstufe I. In : LOG IN, Beilage, 28. Jahrgang (2008), Heft Nr. 150/151.
zitiert: GI, 2008

Helmich, Ulrich: Zeiger.
URL: https://www.u-helmich.de/inf/BlueJ/lexikon/S-Z/Zeiger.html.
zuletzt geprüft: 19.07.2019
zitiert: Helmich

Hesse, Hermann-Günter; Mitter, Wolfgang; Kopp, Botho von [Hrsg.]: Lehr-Lern-Zeit und Lernerfolg aus psychologischer Sicht. In : Die Zeitdimension in der Schule als Gegenstand des Bildungsvergleichs. Köln ; Weimar ; Wien: Böhlau, 1994, S. 143-161.
zitiert: Hesse, 1994

Hubwieser, Peter: Didaktik der Informatik : Grundlagen, Konzepte, Beispiele. Berlin; Heidelberg; New York: Springer-Verlag, 2007.
zitiert: Hubwieser, 2007

Humbert, Ludger; Schubert, Sigrid: Fachliche Orientierung des Informatikunterrichts in der Sekundarstufe II. Didaktik der Informatik Universität Dortmund: Report Nr. 771,
Februar 2002.
URL: https://bscw.ham.nw.schule.de/pub/bscw.cgi/d23846/Uni_Do_cs_ddi_Forschungsbericht_771.pdf.
zuletzt geprüft: 19.07.2019
zitiert: Humbert, 2002

Humbert, Ludger: Didaktik der Informatik – Teil 1. Vorlesungsskript Revision 1.22, Dortmund, 13. November 2003.
URL: https://eldorado.tu-dortmund.de/bitstream/2003/21343/2/teil1.pdf.
zuletzt geprüft: 19.07.2019
zitiert: Humbert, 2003

Klafki, Wolfgang: Didaktische Analyse als Kern der Unterrichtsvorbereitung. In: Die Deutsche Schule. Band 50, Heft 10. Berlin; Hannover; Darmstadt: Schroedel, 1958, Seite 450-471.
zitiert: Klafki, 1958

KMK: Einheitliche Prüfungsanforderungen Informatik, Beschluss der Kultusministerkonferenz vom 01.12.1989 in der Fassung vom 05.02.2004.
zitiert: KMK, 2004

Koch, Wilfried: Professionelles Programmieren von Anfang an mit FreePascal und der freien Entwicklungsumgebung Lazarus. Oberkochen: Oberkochener Medienverlag, 2016.
zitiert: Koch, 2016

Kohls, Christian: Vorlesungsteil Visuelle Programmierung. Fachhochschule Köln, November 2014.
URL: http://www.kohls.de/wp-content/uploads/2014/11/Skriptteil_VisuelleProgrammierung.pdf.
zitiert: Kohls, 2014

Lackes, Richard; Siepermann, Markus: Gabler Wirtschaftslexikon : Syntax einer Programmiersprache.
URL: https://wirtschaftslexikon.gabler.de/definition/syntax-einer-programmiersprache-47018/version-270289.
zuletzt geprüft am 13.07.2019
zitiert: Lackes, Siepermann

Modrow, Eckart: Visuelle Programmierung -oder: Was lernt man aus Syntaxfehlern?
URL: https://subs.emis.de/LNI/Proceedings/Proceedings189/27.pdf.
zuletzt geprüft: 16.07.2019
zitiert: Modrow

Modrow, Eckart; Strecker, Kerstin: Didaktik der Informatik. Berlin: Walter de Gruyter GmbH & Co KG, 2016.
zitiert: Modrow, 2016

Moegling, Klaus: Transparenz beim fächerübergreifenden Lernen – Ein notwendiges Kriterium für die Intensivierung fächerübergreifender Lernprozesse : Transparenz im Unterricht und in der Schule. In : Schulpädagogik heute, 6. Jahrgang (2015), Heft 12.
URL: http://www.schulpaedagogik-heute.de/SHHeft14/03_Praxisartikel/03_19.pdf.
zuletzt geprüft am 19.05.2019
zitiert: Moegling, 2015

Pössel, Markus: Spectrum.de SciLogs : BLOG: RELATIV EINFACH … aber nicht einfacher : Programmieren in der Schule? 19. August 2016.
URL: https://scilogs.spektrum.de/relativ-einfach/programmieren-schule/.
zuletzt geprüft am 13.07.2019
zitiert: Pössel, 2016

Rechenberg, Peter; Pomberger, Gustav: Informatik-Handbuch. 3. Auflage. München: Hanser, 2006.
zitiert: Rechenberg, 2006.

Comenius-Institut [Hrsg.]:Reform der sächsischen Lehrpläne : Fachübergreifender und fächerverbindender Unterricht : Comenius-Institut, 2004,
URL: https://www.sachsen.schule/~nw/tc/files/bg_lp_fachuebergreifender_und_faecherverbindender_unterricht.pdf.
zuletzt geprüft am 13.07.2019
zitiert: Comenius, 2004

Sächsisches Staatsministerium für Kultus und Sport [Hrsg.]: Lehrplan Gymnasium Informatik. Dresden: Saxoprint GmbH, 2011.
zitiert: SMK, 2011

Schubert, Sigrid; Schwill, Andreas; Schwill, Andreas: Didaktik der Informatik. 2. Auflage. Berlin; Heidelberg; New York: Springer-Verlag, 2011.
zitiert: Schwill, 2011

Schwill, Andreas: Programmierstile im Anfangsunterricht,
URL: http://hyfisch.de/didaktik/Forschung/INFOS95.pdf.
zuletzt geprüft am 13.07.2019
zitiert: Schwill

Schwill, Andreas: Programmiersprachen im Informatikunterricht,
URL: http://www.informatikdidaktik.de/Forschung/Schriften/PsimUnterrMatNatTag.pdf.
zuletzt geprüft am 13.07.2019
zitiert: Schwill, 1998

Teriete, Jan: Grundlagen der PHP 7 Programmierung. Version 7.0.2. Nürnberg: Webmaster Akademie Nürnberg GmbH, 2018.
zitiert: Teriete, 2018

Ullenboom, Christian: Java ist auch eine Insel : Das umfassende Handbuch. 9. Auflage. Rheinwerk Computing, 2010.
URL: http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_00_001.htm.
zuletzt geprüft am 13.07.2019
zitiert: Ullenboom, 2010

Wagenknecht, Christian: Programmierparadigmen : Eine Einführung auf der Grundlage von Racket. 2. Auflage. Berlin; Heidelberg; New York: Springer-Verlag, 2016.
zitiert: Wagenknecht, 2016