Qualitätssicherung in der Softwareentwicklung mit formalen Methoden

Originalartikel

Backup

<html> <figure class=„aufmacherbild“></figure> Formale Methoden strukturieren den Entwicklungsprozess von Software, sorgen für nachhaltige Qualitätssicherung und vermeiden so hohe Folgekosten. Seit vielen Jahren steigen die Qualitätsansprüche an Software in breiten, kritischen Anwendungen zum Beispiel bei Maschinensteuerungen, Fahrerassistenzsystemen oder für sichere Cloud- und Blockchain-IT. Dazu gehören Systeme, mit denen Millionen von Menschen täglich in Berührung kommen. Die Kosten für die Qualitätssicherung solcher Anwendungen wachsen mit deren komplexitäts- und veränderungsbedingter Fehleranfälligkeit. Sowohl Kosten als auch Fehler werden nicht selten von geduldigen, unwissenden und machtlosen Kundinnen und Kunden getragen, ohne Haftungsansprüche geltend machen zu können. Aufgrund dessen stellt sich die Frage, ob Softwareentwicklung generell auf stärkere methodische Beine gestellt, im engeren Sinne strenger oder formaler, werden muss? Welche Rolle sollen fachspezifische Formalismen, insbesondere die aus der Softwaretechnik stammenden formalen Methoden, in Informatiklehrplänen an Hochschulen und anderen Ausbildungsstätten spielen? [1] [1] In der international beachteten Referenz ACM/IEEE-CS/AAAI Computer Science Curricula 2023 [2] finden sich beispielsweise nur knappe Lehrplanempfehlungen dazu im Softwaretechnikteil sowie einige Empfehlungen für Vertiefungsvorlesungen im Programmiersprachenteil. Ein wichtiger Grund für diese Überlegungen ist sicherlich, dass Software immer öfter eine sehr viel kritischere Rolle spielt als noch vor zehn Jahren. Ein weiterer Grund ist, dass Softwareentwicklung heute durch leistungsfähige Hochsprachen, Plattformen und Entwicklungsökosysteme auf einer abstrakteren, agileren Ebene stattfindet, und daher auch ein häufig abstrakteres, jedoch präzises Denken gefordert, um korrekte Software zuverlässig zu konstruieren und trotz erforderlicher Änderungen verlässlich in Betrieb zu halten. [2] [3] Die dazu nötigen Kenntnisse haben sich über die Jahrzehnte in den sogenannten formalen

Methoden niedergeschlagen. Eine Ausbildung in Informatik kann aufzeigen, wie solche Methoden in kritischen Sparten der Software- und Systementwicklungspraxis [3] [4] mithilfe von leistungsfähigen Werkzeugen angewendet werden. Gerade weil künstliche Intelligenz (KI) die Programmierung zunehmend automatisiert, sollte die Vermittlung dieser Grundfertigkeiten Kernelement jeder modernen Informatikausbildung sein [4] [5]. Dieses Ziel sollte auch dann im Fokus stehen, wenn sich, vielleicht anders als früher, eine recht breite, heterogen vorgebildete Masse an Menschen für eine Informatikausbildung entscheidet. <h3 class=„subheading“ id=„nav_formale0“>Formale Methoden sind nützlich</h3> Der Begriff der formalen Methode existiert in der Informationstechnik seit Jahrzehnten und wird verschieden ausgelegt. Forschende meinen damit <ol class=„rtelist rtelist–ordered“>
die Ad-hoc-Formalisierung eines Programmierproblems mittels formaler Logik, Mengenlehre und anderen Basiswerkzeugen der Mathematik, um das Problem besser zu verstehen,
die durchgängige Formalisierung einer Programmiersprache für die teil- oder vollautomatische Prüfung (formale Deduktion) von Programmeigenschaften,
ein mathematisch fundiertes Werkzeug für die präzise Softwaremodellierung und Modellprüfung. Andere Wissenschaftler und Wissenschaftlerinnen erweitern oder kombinieren diese drei Ansätze zu einer umfassenden Ingenieursmethode. Der Allgemeinheit halber soll hier die dritte Charakterisierung eine Rolle spielen. Danach kann man eine formale Methode als eine mathematisch und logisch fundierte, meist werkzeuggestützte Herangehensweise an die Konstruktion und Analyse einer laufenden Software, einer Anforderungsspezifikation, eines Stücks Quelltext oder eines Modells davon verstehen. Herzstück der Methodenanwendung ist die zweckspezifische Abstraktion und das Herausarbeiten der relevanten Details und des Kontexts einer Softwareanwendung mithilfe einer möglichst übersichtlichen Notation. Es braucht formale Methoden dann, wenn die Anforderungen an die Korrektheit und Zuverlässigkeit einer Software besonders hoch sind, etwa die Steuerung eines Kraftwerks oder einer besonders risikobehafteten Maschine, die Abwicklung eines Online-Bezahlvorgangs oder die Übertragung einer elektronischen Patientenakte. Durch eine strenge Vorgehensweise können Entwicklerinnen und Entwickler sowohl die korrekte Konstruktion als auch die Absicherung der Software gut nachvollziehbar dokumentieren und kommunizieren. Das spielt ebenfalls eine wichtige Rolle, beispielsweise für eine TÜV- oder ISO-Zertifizierung sowie für die Klärung von Haftungsfragen vor Gericht. Ein aktuelles Anwendungsbeispiel ist ferner der Nachweis, dass das von Fahrerassistenzsoftware ausgehende Unfallrisiko unter einer tolerierbaren Schwelle liegt. In der Tat empfehlen einschlägige Standards (z.B. DO-178C, EN 5012x, IEC 61508, ISO 26262) den Einsatz formaler Ansätze in besonders riskanten Szenarien. Damit hätten moderne formale Methoden mit ausgereiften Werkzeugen heute nicht nur das Potenzial, sondern auch die legale Möglichkeit, bei Produkthaftungsfragen als Referenz für den Stand der Qualitätssicherungspraxis zu dienen. Allein schon deshalb gebührt ihnen ein fester Platz in den Lehrplänen der Ausbildungsstätten. <figure class=„a-inline-image a-u-inline“> <img alt=„“ class=„legacy-img c1“ height=„391“ sizes=„“ src=„https://heise.cloudimg.io/width/696/q85.png-lossy-85.webp-lossy-85.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb1-7bbb18868ee4ad79.png“ srcset=„https://heise.cloudimg.io/width/336/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb1-7bbb18868ee4ad79.png 336w, https://heise.cloudimg.io/width/1008/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb1-7bbb18868ee4ad79.png 1008w,

https://heise.cloudimg.io/width/696/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb1-7bbb18868ee4ad79.png 696w,
https://heise.cloudimg.io/width/1392/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb1-7bbb18868ee4ad79.png 1392w“ width=„696“ referrerpolicy=„no-referrer“ /> <figcaption class=„a-caption“>Hierarchischer endlicher Automat zur prägnanten Beschreibung von Abläufen (Abb. 1).(Bild: Mario Gleirscher)</figcaption></figure><h3 class=„subheading“ id=„nav_klarstellung1“>Klarstellung mit Formalismen</h3> So wie in anderen Technikwissenschaften spielen Formalismen und auch in der Informatik eine vielschichtige Rolle. Über gut ein Jahrhundert lang hat die Informatikforschung viele solcher Formalismen genutzt. Vielleicht ist es die im Vergleich zu anderen Ingenieursdisziplinen und Wissenschaften schlechte Greifbarkeit kreierter Dinge (Software), die neue Modellierungs- und Darstellungsversuche in Formalismen antreibt (z.B. mit Automaten und Zustandsübergangsdiagrammen, siehe Abbildungen 1 und 2)? Vielleicht ist es diese schlechte Greifbarkeit auch, die – ähnlich wie in der theoretischen Physik – neue, abstraktere Anwendungen und Erweiterungen der Mathematik benötigte. Um die Grundlagen zu vermitteln, aber auch um neue Erkenntnisse zu gewinnen, benötigt man jedenfalls häufig: <ul class=„rtelist rtelist–unordered“>
abstrakte Maschinenmodelle und Mengentheorie für Komplexitätsuntersuchungen von Problemen und Algorithmen,
relationale Algebra, formale Sprachen und Automatentheorie zur genauen Modellierung und Beschreibung der Funktionsweise von Datenbanken, Systemsoftware und Programmiersprachen oder
algebraische Datentypen, Zustandsmaschinen, temporale Logik und Kategorientheorie für die schrittweise Modellbildung im Softwareentwurf und in der Programmprüfung. Eine breite Erkenntnis vieler Technikwissenschaften ist, dass ohne die Anwendung von Formalismen keine kontrollierte Annäherung an einen expliziten, konsensfähigen und stabilen Wissens- und Methodenkorpus erfolgen kann. <figure class=„a-inline-image a-u-inline“> <img alt=„“ class=„legacy-img c1“ height=„391“ sizes=„“ src=„https://heise.cloudimg.io/width/696/q85.png-lossy-85.webp-lossy-85.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb2-6e35e2bc34e86208.png“
srcset=„https://heise.cloudimg.io/width/336/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb2-6e35e2bc34e86208.png 336w,
https://heise.cloudimg.io/width/1008/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb2-6e35e2bc34e86208.png 1008w,
https://heise.cloudimg.io/width/696/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb2-6e35e2bc34e86208.png 696w,
https://heise.cloudimg.io/width/1392/q70.png-lossy-70.webp-lossy-70.foil1/_www-heise-de_/imgs/18/4/3/2/1/1/9/5/abb2-6e35e2bc34e86208.png 1392w“ width=„696“ referrerpolicy=„no-referrer“ /> <figcaption class=„a-caption“>SysML State Chart, ausdrucksmächtiges Mittel einer standardisierten und gut formalisierbaren Software-Modellierung (Abb. 2).(Bild: Mario Gleirscher)</figcaption></figure><h3 class=„subheading“ id=„nav_mehr_als_nur2“>Mehr als nur Formalismen</h3> Wenn es um die Konstruktion komplexer kritischer Software geht, also um unübersichtliche, schwer greifbare oder wenig anschauliche Gegenstände (Datenstrukturen) und Sachverhalte (Algorithmen), müssen die Konstruktionstechniken besonders hohe Anforderungen an eine präzise <ul class=„rtelist rtelist–unordered“>
Wissensdarstellung (z.B. konsistente, formale Bedeutungslehre),
Wissenshandhabung (z.B. Abbildung intuitiver Konstruktions- und Analyseschritte) und
Wissenskommunikation (z.B. Zweck spezifische, konsistente, übersichtliche Darstellungen) erfüllen. Neben den vertrauten Programmkonstrukten und Programmierrichtlinien bieten formale Methoden zusätzliche Abstraktionstechniken, deren Nutzen in jahrelanger Forschung stets zunahm. Im Sinne einer Ingenieursmethodik mit Leitlinien und Werkzeugen lassen sich beispielsweise, ohne einen Anspruch auf Vollständigkeit oder Aktualität der Auflistung, folgende Ansätze nennen: <a

href=„https://doi.org/10.1007/978-3-642-18216-7“ rel=„external noopener“ target=„_blank“>ASM [6], B [5] [7], CASL [8], Focus [9], LOTOS [10] (ISO-standardisiert), RAISE [11], TLA+ [12], UNITY [6] [13], VDM [14], SCR [15] und Z [16] (ISO-standardisiert). Derartige Methoden wurden je nach Fall ab etwa Anfang der 1980er teils von mehreren Forschenden gemeinsam entwickelt und teils in verschiedene Richtungen verfeinert. Sie bedienen sich einer Reihe aus der formalen Logik, Mathematik und der theoretischen Informatik stammender Konzepte – Prädikatenlogik, relationale Algebra, Funktionen, Automatentheorie, formale Sprachen, Prozessalgebra, temporale Logik und Modelltheorie – und formalen Schließtechniken wie Formen des deduktiven Schließens, die Arbeit mit Invarianten und der Modellprüfung. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Leicht verarbeitbare Textdarstellung eines diskreten hybriden Automaten in HyTech, hilfreiches Mittel zur formalen Modellierung eingebetteter Systeme (Abb. 3).(Bild: Mario Gleirscher)</figcaption></figure> Entscheidend für viele formale Methoden sind aber die methodischen Grundelemente [7] [17] sowie die am Nutzer orientierten Leitlinien, die sich in den Notationen und der praktischen Handhabung dieser Formalismen ausprägen (siehe Abbildungen 3 und 4). Agile Vorgehensweisen der Softwareentwicklung beispielsweise erfordern einen präzisen Umgang mit Änderungen im Kontext der kontinuierlichen Auslieferung [2] [18]. Unter

anderem der Formal Methods Europe Verein bietet neben einer Methodenauswahlhilfe einen Literaturüberblick zu den oben genannten Ansätzen und Werkzeugen. Hinzu kommen eine umfangreiche Selektion von Lehr- und Trainingsmaterialien sowie regelmäßige Seminare zum Kennenlernen formaler Methoden und zum Austausch zwischen Forschenden und Praktizierenden – zwischen Akademie und Industrie [8] [19]. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Grafische Darstellung des Automaten (Abb. 4).(Bild: Mario Gleirscher)</figcaption></figure> Es ist wichtig zu verstehen, dass es sich bei den heute weitverbreiteten Modellierungssprachen wie UML oder SysML, aber auch bei reinen Formalismen (z.B. Prädikatenlogik) oder heute weniger genutzten Sprachfamilien (IDEF, STEP uvm.) weder um formale Methoden noch im engeren Sinne überhaupt um Methoden handelt. Ein Formalismus oder eine Programmier- oder Modellierungssprache wird nicht etwa deshalb zu einer vollwertigen Methode, weil deren Notation formal definiert ist oder deren Sprachkonstrukte eine gewisse Vorgehensweise suggerieren (siehe Abbildung 5). Hingegen muss man modernen Hochsprachen einen methodischen Gehalt zusprechen, etwa wenn es um die Umsetzung bewährter Konstruktionsprinzipien: Typsicherheit, Kapselung, Zusicherungen (z.B. Assertions in Java), Diagrammtypen, Kompositionsbeziehungen (u.a. in SysML) usw. geht. Der Anspruch formaler Methoden zielt jedoch darüber hinaus auf eine umfassende, formal fundierte Unterstützung korrekter Programmentwicklung [7] [20], in gewissem Sinne auch eine Erweiterung testgetriebener und prototypbasierter Entwicklung. Eine zunehmend gemeisterte Herausforderung ist die Integration gut automatisierbarer, formaler Methoden in moderne, agile Softwareentwicklungsprozesse [2] [21]. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>SysML Blockdiagramm zur einfachen, dennoch gut formalisierbaren Darstellung der Architektur eines verteilten Softwaresystems (Abb.

5).(Bild: Mario Gleirscher)</figcaption></figure><h3 class=„subheading“ id=„nav_bisherige3“>Bisherige Nutzung formaler Methoden</h3> Formale Methoden im engeren Sinne entstammen der Softwaretechnikforschung und wurden sukzessive in der Informatiklehre zur Aneignung von Forschungserkenntnissen eingesetzt. Später übernahm die industrielle Software- und Systementwicklung formale Methoden, meist jedoch wie erwähnt in hochkritischen Anwendungen oder zu praxisorientierten Forschungszwecken [9] [22]. Zunehmend Anwendung finden die vergleichsweise gut automatisierbaren Techniken der statischen Analyse (z.B. abstrakte Interpretation), Typsysteme und die Modellprüfung (Model Checking). Weitere erfolgreiche Bestrebungen, formale Methoden über die modellbasierte oder -getriebene Entwicklung (oft mittels UML, SysML oder domänenspezifischen Sprachen, siehe Abbildungen 2 und 5) in die moderne Softwareentwicklung einzubringen, gibt es bereits in Branchen wie Automotive, Avionik, Bahn oder Fertigungsautomatisierung. Hierbei stehen vielfältige Vorteile (bessere Koordination von Entwicklungsabteilungen, Wiederverwendung, adäquate Abstraktionen) auch einigen Nachteilen gegenüber (eingeschränkte Sprachnutzung, indirekte Code-Optimierung, stärkere Abhängigkeit von Werkzeugen). Zu den erfolgreichen Beispielen für die Nutzung formaler Methoden gehören: <ul class=„rtelist rtelist–unordered“>
die Prüfung von nur teuer aktualisierbaren Hardwareschaltungen im Sinne des Right-the-first-time-Prinzips (Prozessorverifikation bei ARM, Infineon oder Intel),
die Härtung von häufig mit Angriffen bedrohten, sicherheitskritischen Systemfunktionen (Betriebssystemkerne, Zugriffskontroll- und Ressourcentrennmechanismen in Rechenzentren),
die Protokollverifikation in naturgemäß komplexen, trägen oder heterogenen, verteilten Anwendungen mit breiten, kostenintensiven Fehlerkonsequenzen (Telekommunikation, Datenbank- und Blockchain-Technologie), und
in der Regelungsinformatik, die Prüfung akut risikobehafteter Regelungssoftware (Steuerungen für Nutzfahrzeuge oder Kernkraftwerke) im Rahmen einer haftungsrechtlich gebotenen Zertifizierung. Zur gezielten Unterstützung dieser Prüfungsaufgaben kommen Formalismen und spezifische formale Methoden zur Definition der in diesen Bereichen relevanten Abstraktionen und der damit verwendeten Modellierungs- und Programmiersprachen zum Einsatz. Maßgeblich für eine erfolgreiche Anwendung formaler Methodik ist das Verständnis der nötigen Grundlagen, weitgehend unabhängig davon, ob eine Methodenanwendung vollständig oder teilweise automatisiert erfolgen kann oder nicht. Insgesamt konnte sich bisher keine Methode breit etablieren. Dafür gibt es viele Gründe: mangelnde Lehre, unausgewogene Lehrpläne, Fehleinschätzungen der Lehrenden zu langfristig relevanten Inhalten, unzureichendes Interesse oder fehlende Grundkompetenzen bei den Auszubildenden [10] [23], ungeschultes Personal in Unternehmen, Versäumnisse in der Methodenentwicklung beim Management dieser Unternehmen, Fehleinschätzungen in der Kosten/Nutzen-Bewertung neuer Methoden und Technologien, unzureichende Skalierbarkeit oder Praxisnähe der angebotenen Methoden und fehlende Abstraktionstechniken [11, 12] [24], verbesserungswürdige Werkzeugunterstützung, kurzfristige wirtschaftliche Interessen der Werkzeugentwickler, hohe finanzielle Leidensfähigkeit von Kunden oder Nutzerinnen gegenüber Softwareproblemen. Künftige Softwareentwicklung profitiert von formalen Methoden Zwei interessante Entwicklungen weisen darauf hin, dass

werkzeuggestützte, formale Methoden – und die damit einhergehende, flexible Fähigkeit der Abstraktion – in Zukunft eine stärkere Rolle in vielen Domänen spielen können. Zum einen wird mit der steigenden Bedeutung korrekter, zuverlässiger, jedoch zunehmend komplexer Software in vielen Gesellschaftsbereichen auch die Anwendung stringenter Methodik in der Systementwicklung und Qualitätssicherung steigen. Fehler und Lücken sind zunehmend schwer oder teuer aufzudecken und zu beheben. Damit dürfte die Anwendung formaler Methoden zur Fehlervermeidung in frühen oder korrigierenden Entwicklungsphasen häufiger notwendig sein (siehe Abbildung 6) oder gar zum Selbstverständnis werden. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Anzahl von Befragten, welche formale Methoden in einem oder mehreren Anwendungsbereichen genutzt haben und/oder nutzen wollen (Abb. 6).(Bild: Finney, K. 1996 [9])</figcaption></figure> Zum anderen verschiebt die zunehmende Automatisierung vieler bisher manueller Aufgaben durch Techniken der künstlichen Intelligenz viele Berufsfelder in kreativere oder anspruchsvollere Tätigkeiten. Man wird vielerorts quasi vom Problemlöser zum Problemspezifizierer, muss keine Einzellösungen Zeile für Zeile erarbeiten, sondern Lösungsräume einschränken und Lösungsvorschläge prüfen oder zuverlässig anpassen. Oft wird auch die Programmreparatur oder die Fehlerbehebung automatisiert erfolgen. Für all dies müssen gewisse Abstraktionen erdacht und genutzt werden. Gerade das gehört zu den Kernfähigkeiten formaler Methoden, insbesondere der Spezialdisziplin des verfeinerungsbasierten Entwurfs [5] [25]. In der Softwareentwicklung können moderne, Suche-basierte Entwicklung, Programmsynthese und künstliche Intelligenzwerkzeuge wie ChatGPT oder GitHub Copilot durchaus dazu führen, dass Ingenieurinnen und Entwickler zunehmend ihre Stärken in der Anwendungsanalyse, der Domänenabstraktion, der Problemspezifikation sowie der Prüfung generierter Software [13] [26] ausleben. <h3 class=„subheading“ id=„nav_fazit_4“>Fazit</h3> Aus den beiden und weiteren Gründen wird es wichtig sein, auf diesen neuartigen Methodenbedarf oder den Bedarf an neuen Herangehensweisen, sowohl in der Forschung als auch in der Informatiklehre zu reagieren und so die nächsten Kohorten von Softwareentwicklern auf neue Herausforderungen vorzubereiten [11] [27]. Moderne formale Methoden könnten also nicht wie bisher hauptsächlich als Denkwerkzeuge für die Forschung und Grundausbildung dienen, sondern einen neuen akzentuierten Zweig im Lehrplan der Informatikausbildung besetzen, der

Problemspezifikation und -verfeinerung, und das schon ab der Sekundarstufe [1] [28]. So könnten die oben angesprochenen Fähigkeiten mittels einer Methodenlehre an praktizierende, KI-nutzende Softwareingenieurinnen und -ingenieure der Zukunft besonders effektiv weitergegeben werden. KI-basierte Softwareentwicklung mithilfe formaler Methoden könnte einen nützlichen Paradigmenwechsel einläuten und moderne Qualitätsansprüche an Software besser adressieren. Diesen Paradigmenwechsel gilt es bildungspolitisch auf allen Ebenen, in der Sekundarstufe, in der Ausbildung an Hochschulen und in der Forschung, zu unterstützen. Ferner ist die Einsicht nützlich, dass eine an formalen Grundlagen orientierte universelle Ausbildung jemanden auch darin befähigt, sich änderndes, praktisches Wissen selbst zu erschließen, während eine Orientierung der Ausbildung an kurzfristig aktuellen Themen und Technologien derart wichtige, berufliche Fähigkeiten nicht unbedingt begünstigt. Mario Gleirscher
forscht und lehrt derzeit im Fachbereich Mathematik und Informatik an der Universität Bremen zum Themenschwerpunkt Verifikation und Synthese von Maschinensteuerungen in der Robotik und anderen hoch automatisierten technischen Prozessen. Er interessiert sich auch für die industrielle Anwendung formaler Methodik und ist unter anderem stellvertretender Sprecher der Fachgruppe FoMSESS [29] (Formale Methoden und Software Engineering für sichere Systeme) in der Gesellschaft für Informatik e.V. Danksagung: Mein herzlicher Dank für sehr hilfreiches Feedback geht an Clemens Lanthaler. <h5 id=„nav_literatur_5“>Literatur</h5> [30][1] Broy, M.; Brucker, A.; Fantechi, A.; Gleirscher, M.; Havelund, K.; Jones, C.; Kuppe, M.; Mendes, A.; Platzer, A.; Ringert, J. O. und Sullivan, A. (2023). Does Every Computer Scientist Need to Know Formal Methods?, Form. Asp. Comput. In Druck. [31][2] O'Hearn, P. W. (2018). Continuous Reasoning. In: Dawar, A. & Grädel, E. (Ed.), LICS, 33rd Ann. Symp., ACM Press. DOI: https://doi.org/10.1145/3209108.3209109 [32] [33][3] Jones, C. B. und Thomas, M. (2022). The Development and Deployment of Formal Methods in the UK, Form. Asp. Comput. 34 : 1-21. DOI: https://doi.org/10.1145/3522577 [34] [35][4] Welsh, M. (2022). The End of Programming, Commun. ACM 66 : 34-35. DOI: https://doi.org/10.1145/3570220 [36] [37][5] Abrial, J.-R., 2010. Modeling in Event-B. Cambridge UP, Cambridge. ISBN: 9780521895569 [38][6] Chandy, K. M.; Misra, J.(1989). Parallel Program Design: A Foundation. Addison Wesley: Reading, MA. ISBN: 0-201-05866-9 [39][7] Brinksma, E. (1991). What is the Method in Formal Methods?. In: Parker, K. R. & Rose, G. A. (Ed.), FORTE, 4th Int. Conf., Elsevier. DOI: https://doi.org/10.1016/b978-0-444-89402-1.50012-6 [40]. [41][8] Formal Methods Europe (2023). Choosing and Learning a Formal Method, https://www.fmeurope.org/industry/ [42], <a

href=„https://www.fmeurope.org/choosingaformalmethod/“ rel=„external noopener“ target=„_blank“>https://www.fmeurope.org/choosingaformalmethod/ [43], https://fme-teaching.github.io/courses/ [44]. [45][9] Gleirscher, M. und Marmsoler, D. (2020). Formal Methods in Dependable Systems Engineering: A Survey of Professionals from Europe and North America, Empir. Softw. Eng. 25 : 4473-4546. DOI: https://doi.org/10.1007/s10664-020-09836-5 [46] [47][10] Finney, K. (1996). Mathematical notation in formal specification: too difficult for the masses?, IEEE Trans. Software Eng. 22 : 158-159. DOI: https://doi.org/10.1109/32.485225 [48] [49][11] Gleirscher, M.; van de Pol, J. und Woodcock, J. (2023). A Manifesto for Applicable Formal Methods, Softw. Syst. Model. 0 : 1-17. DOI: https://doi.org/10.1007/s10270-023-01124-2 [50] [51][12] Gleirscher, M.; Foster, S. und Woodcock, J. (2019). New Opportunities for Integrated Formal Methods, ACM Comput. Surv. 52 : 117:1-117:36. DOI: https://doi.org/10.1145/3357231 [52] [53][13] Meyer, B. (2023). AI Does Not Help Programmers, BLOG@CACM, https://cacm.acm.org/blogs/blog-cacm/273577-ai-does-not-help-programmers [54]. () <hr /> URL dieses Artikels:
<small>

```
https://www.heise.de/-9339463
```

</small> Links in diesem Artikel:
<small>

```
<strong>[1]</strong> #anchor_1
```

</small>
<small>

```
<strong>[2]</strong> https://csed.acm.org/programming-languages/
```

</small>
<small>

```
<strong>[3]</strong> #anchor_2
```

</small>
<small>

```
<strong>[4]</strong> #anchor_3
```

</small>
<small>

[5] #anchor_4

</small>
<small>

[6] https://doi.org/10.1007/978-3-642-18216-7

</small>
<small>

[7] #anchor_5

</small>
<small>

[8] https://doi.org/10.1007/b11968

</small>
<small>

[9] https://doi.org/10.1007/978-1-4613-0091-5

</small>
<small>

[10] https://www.iso.org/standard/16258.html

</small>
<small>

[11] https://raisetools.github.io/

</small>
<small>

[12] https://doi.org/10.1007/978-1-4842-3829-5

</small>
<small>

[13] #anchor_6

</small>
<small>

[14] https://doi.org/10.1007/3-540-08766-4

</small>
<small>

[15] https://doi.org/10.1109/cmpass.1995.521891

</small>
<small>

[16] https://www.iso.org/standard/21573.html

</small>
<small>

[17] #anchor_7

</small>
<small>

[18] #anchor_2

</small>
<small>

[19] #anchor_8

</small>
<small>

[20] #anchor_7

</small>
<small>

[21] #anchor_2

</small>
<small>

[22] #anchor_9

</small>
<small>

[23] #anchor_10

</small>
<small>

[24] #anchor_11

</small>
<small>

[25] #anchor_5

</small>
<small>

[26] #anchor_13

</small>
<small>

[27] #anchor_11

</small>
<small>

[28] #anchor_1

</small>
<small>

[29] https://fg-fomsess.gi.de/

</small>
<small>

[30]

</small>
<small>

[31]

</small>
<small>

[32] https://doi.org/10.1145/3209108.3209109

</small>
<small>

[33]

</small>
<small>

[34] https://doi.org/10.1145/3522577

</small>
<small>

[35]

</small>
<small>

[36] https://doi.org/10.1145/3570220

</small>
<small>

[37]

</small>
<small>

[38]

</small>
<small>

[39]

</small>
<small>

[40] https://doi.org/10.1016/b978-0-444-89402-1.50012-6

</small>
<small>

[41]

</small>
<small>

[42] https://www.fmeurope.org/industry/

</small>
<small>

[43] https://www.fmeurope.org/choosingaformalmethod/

</small>
<small>

[44] https://fme-teaching.github.io/courses/

</small>
<small>

[45]

</small>
<small>

[46] https://doi.org/10.1007/s10664-020-09836-5

</small>
<small>

[47]

</small>
<small>

[48] https://doi.org/10.1109/32.485225

</small>
<small>

[49]

</small>
<small>

[50] https://doi.org/10.1007/s10270-023-01124-2

</small>
<small>

[51]

</small>
<small>

[52] https://doi.org/10.1145/3357231

</small>
<small>

[53]

</small>
<small>

[54] https://cacm.acm.org/blogs/blog-cacm/273577-ai-does-not-help-programmers

</small>
<small>

[55] mailto:who@heise.de

</small>
 Copyright © 2023 Heise Medien </html>