Vertrauen in KI-basierte Systeme schaffen

Ina Schieferdecker [1,2,3]

1 Weizenbaum-Institut für die Vernetzte Gesellschaft

2 Fraunhofer FOKUS

3 Technische Universität Berlin

Im Zuge des digitalen Wandels werden zunehmend Software-basierte Systeme, die auch Methoden der Künstlichen Intelligenz (KI)[4] nutzen, mit menschlichen Aktivitäten abgestimmt. Getrieben von Daten, die von Menschen erzeugt, aggregiert und / oder aufbereitet werden, arbeiten solche, auch kurz genannt KI-basierten Systeme immer enger mit uns Menschen in Echtzeit zusammen. Die Suche nach den dafür benötigten passenden und leistungsfähigen Automatismen mittels KI ist ebenso eine Suche nach den Erwartungen und Anforderungen, die an solche Systeme zu legen sind und wie diese im Entwurf, in der Entwicklung und im Betrieb abgesichert werden können. Vertrauenswürdigkeit als eine der wesentlichen Grundlagen zur Akzeptanz KI-basierter Systeme muss dazu über Korrektheit sowie Zuverlässigkeit, Leistungsfähigkeit sowie Skalierbarkeit, Sicherheit sowie Datenschutz als auch Nachhaltigkeit solcher Systeme erreicht werden. Dieser Artikel bietet dazu eine Reflektion des Standes des Software Engineerings als Disziplin zur Entwicklung vertrauenswürdiger Software und zudem Parallelen als auch neue Herausforderungen bei der Entwicklung KI-basierter Systeme. Der Artikel ist eine Weiterentwicklung von (Schieferdecker, Großmann, & Schneider, 2019; Schieferdecker & Ritter, 2019).

Einleitung

Die Menschheit hat sich über Jahrtausende dienliche und dienende Automatismen geschaffen, mit denen vordefinierte Abläufe, sogenannte Algorithmen, durch Maschinen selbständig realisiert werden. Maschinen basieren dabei auf mechanischen, elektronischen oder digitalen Methoden der Automatisierung und werden mittlerweile in bald allen Lebens- und Arbeitsbereichen wie Produktion, Landwirtschaft, Bau, Transport oder auch der Unterhaltung genutzt. Mit Informations- und Kommunikationstechnologien als Fundament und der zunehmenden Digitalisierung und Vernetzung

4 Künstliche Intelligenz (KI) (WBGU, 2019: 482) bezeichnet „eine Disziplin innerhalb der Informatik, die sich mit der Entwicklung von Softwaresystemen befasst, welche Funktionen bereitstellen, deren Ausführung das erfordert, was typischerweise mit dem Wort Intelligenz bezeichnet wird“ (Burgard, 2018). Ebenso wie bei „Intelligenz“ gibt es bis heute keine einheitliche Definition von Künstlicher Intelligenz, sondern plurale Verständnisse. Grundsätzlich zeichnen sich intelligente Systeme durch die Fähigkeit aus, (teilweise) selbstständig und effizient Probleme zu lösen (Mainzer, 2016: 2).

WBGU – Wissenschaftlicher Beirat der Bundesregierung Globale Umweltveränderungen (2019): Unsere gemeinsame digitale Zukunft. Berlin: WBGU.

Burgard, W. (2018): Künstliche Intelligenz, Externe Expertise für das WBGU-Hauptgutachten „Unsere gemeinsame digitale Zukunft“, Berlin: WBGU.

Mainzer, K. (2016): Künstliche Intelligenz – Wann übernehmen die Maschinen? Heidelberg, Berlin: Springer.

der Maschinen, nehmen ihre Fähigkeiten bezüglich Kognition und Autonomie zu (Fromhold-Eisebith et al., 2019), so dass sich Fragen wie: Sind die Maschinen zuverlässig? Kann man den Automatismen der Maschinen vertrauen? Sind die Maschinen in der Lage, auch unerwartete kritische Situationen zu bewältigen? Welche Entscheidungshoheit sollte beim Menschen verbleiben? stellen. Die Dringlichkeit der Beantwortung dieser Fragen stellt sich auch vor dem Wissen, dass digitalisierte Automatisierung zu unerwünschten soziotechnischen Effekten führen können und bereits mit teils katastrophalen Auswirkungen geführt haben. Erinnert sei an dieser Stelle an die Explosion der Ariane-5-Rakete 1996 (Dowson, 1997) oder aber an jüngste Vorfälle beim automatisierten Fahren[5] oder in der Luftfahrt,[6] die bei der traurigen Bilanz der «Software-Toten" mitzuzählen sind.

Software an sich ist ein Komposit von Programmen und Daten und umfasst zudem Artefakte wie Spezifikationen, Dokumentationen, Nutzungshinweise sowie Prüfspezifikationen, Prüfdaten und / oder Zertifikate. Dabei ist Software im Prinzip eine Realisierung von Algorithmen durch Programme und Daten. Mit der Ausführung der Programme werden die durch sie realisierten Algorithmen auf (Anwendungs-)Daten angewendet und wirken über die berechneten Ergebnisse (wiederum Daten) auf Ein- und Ausgabegeräte, Sensorik, Aktuatorik und / oder Robotik. Ein sehr einfaches Beispiel ist das Sortieren von Zahlen, für die es verschiedene Sortieralgorithmen (wie Bubble Sort, Binary Tree Sort oder Bucket Sort) gibt, die mittels vielfältiger Programmbibliotheken (typischerweise gehören Sortieralgorithmen zu den Standardbibliotheken der Programmiersprachen) realisiert sind, auf Zahlen, die beispielsweise Kredite, Studienergebnisse, Preise oder Laufzeiten darstellen, angewendet werden sowie im Ergebnis als geordnete Listen auf einem Monitor angezeigt oder in einer anderen Software-Funktion weiterverarbeitet werden.

Software entfaltet ihre Wirkung während ihrer Ausführung, so dass die Wirkungen zudem zu der Ausführungsumgebung sowie den Konfigurationen und konkreten Nutzungsszenarien der Software abhängt. Für die Ausführung von Software werden des Weiteren Werkzeuge wie Compiler/ Interpreter, Laufzeitumgebungen der verwendeten Program-

5 Bei tödlichen Unfällen eines Elektrofahrzeugs von Tesla im Mai 2016 als auch im März 2019, bei dem das Fahrzeug jeweils unter einen die Straße querenden Lastwagen-Anhänger fuhr, waren jeweils die Autopiloten aktiviert.

6 Die Maneuvering Characteristics Augmentation System (MCAS)-Software der Boeing 737 Max wird für die Abstürze im März 2019 bei Ethiopian Airlines und im Oktober 2019 bei Lion Air, bei denen 346 Menschen ums Leben gekommen sind, als Ursache gesehen. Sie soll die Nase der Boeing nach unten gedrückt haben, ohne dass die Besatzung in der Lage war, die Fehlsteuerung zu korrigieren.

miersprache, Betriebssysteme und Treiber der verwendeten Hardware als auch Netzwerkkomponenten genutzt. Zusammen ergibt sich so ein komplexes Gefüge zwischen Anbietern, Entwicklern, Betreibern und Nutzenden von Software-basierten Systemen einerseits und den sozio-technischen Komponenten in Software, Hardware und Infrastruktur andererseits.

Dabei führte fehlerbehaftete Software schon in den 1960er bei viel einfacheren Software-Konstellationen (Software in der Größenordnung von KBytes sowie rein sequentiell programmiert ohne Parallelität oder Vernetzung) zur Software-Krise (Randell, 1979) und dem Aufruf, in der Informatik das Gebiet des Software Engineerings zur Konstruktion und Absicherung von Software mit dazu passenden für Konzepten, Methoden, Technologien und Werkzeugen zu entwickeln (Schieferdecker & Ritter, 2019; Wirth, 2008). Seither hat sich Software Engineering als Disziplin in der Informatik etabliert und sich vom maschinennahen Software-Paradigma über das funktionale, objekt-orientierte, komponentenbasierte bis hin zum modellbasierten Software-Paradigma entwickelt. Trotz des gewachsenen Verständnisses und umfangreichen Methodenkastens werden in regelmäßigen Abständen neue Software-Krisen beschrieben (Fitzgerald, 2012; Gibbs, 1994). Und noch heute stellt der Entwurf, die Realisierung und die Absicherung von Software entlang der verschiedenen Dimensionen von Software-Qualität (Miguel, Mauricio, & Rodríguez, 2014), die beispielsweise wie Sicherheit und Leistungsfähigkeit oder Bedienfreundlichkeit im Widerstreit zueinander stehen können, eine Herausforderung dar (Breu, Kuntzmann-Combelles, & Felderer, 2014).

In der Tat gibt es Software in vielen Arten und Varianten und keine in der Wissenschaft akzeptierte Software-Taxonomie. So ist es scheinbar schwer, die Eigenschaften von Software im Allgemeinen zu erfassen, und dennoch gibt es ein langjähriges und immer noch wachsendes Verständnis dafür, wie Softwarequalität[7] zu erfassen und bewerten ist. Insbesondere die ISO 25010 enthält einen Katalog an Qualitätsanforderungen an Software-basierte Systeme (Gordieiev, Kharchenko, Fominykh, & Sklyar, 2014), die aggregiert in der Wissenschaft und Wirtschaft etablierte Software-Qualitätsmodelle wie von Boehm (Boehm, Brown, & Lipow, 1976),

7 Software-Qualität bezeichnet den Grad, in dem ein software-basiertes System, die Kundenerwartungen und -bedürfnisse erfüllt. So umfasst sie alle Merkmale eines Software-basierten Systems, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.

FURPS (Boehm et al., 1976), IEEE (IEEE, 1993) oder QMOOD (Hyatt & Rosenberg, 1996) zusammenfassen.

Vertrauenswürdigkeit von KI-basierten Systemen

Mit Blick auf KI als zunehmend relevante Technologie stellen sich wie bei Software und Software-basierten Systemen vergleichbare Fragen zur Qualität, Verlässlichkeit und Vertrauenswürdigkeit als auch Fragen nach den Relationen zwischen KI und Software sowie zwischen der Entwicklung von KI-basierten Systemen und von Software-basierten Systemen.

Kurz gesagt: KI wird als Software (und ggf. mit spezieller Hardware) realisiert. Und in bald jeder Software wird eine KI-Methode genutzt, auch wenn sie nicht als solche erkennbar ist oder erkannt wird. Generell wird an symbolischer (regelbasierter) KI, an sub-symbolischer (statistischer) KI und jüngstens an Mischformen beider gearbeitet (Buchanan, 2005; LeCun, Bengio, & Hinton, 2015; Norman, 1991). Bei symbolischer KI werden Wissen und Zusammenhänge zwischen Teilen des Wissens formal repräsentiert, um logische Schlüsse ziehen zu können. Bei sub-symbolische KI werden Zusammenhänge aus Merkmalen von großen Datenmengen gezogen. In aktuellen hybriden Methoden werden geschickte Kombinationen beider Ansätze genutzt. Am Ende bietet KI ein Methodenspektrum zur Analyse und / oder Herleitung von Wissen.

Auch wenn KI immer auch Software »ist«, gilt jedoch nur auf einen ersten Blick, dass sie mit »konventionellen« Technologien des Software Engineerings entwickelt und abgesichert werden kann. Einerseits stellen die Techniken rund um die verwendeten großen Datenmengen und ihre statistischen Auswertungen neue Anforderungen an die Definition von Qualität – beispielsweise: Wann ist die Genauigkeit einer Klassifikation akzeptabel? Was bedeutet »akzeptabel« in welchen Kontexten? – und deren Herstellung und Absicherung. Andererseits geht es eben nicht nur um technische Anforderungen, sondern ebenso – wie bei Software-basierten Systemen – um sozio-technische Anforderungen an die Wirkweise KI-basierter Systeme.

So definieren beispielsweise die Algo.Rules (Bertelsmann-Stiftung & iRights.Lab, 2019) einen Kriterienkatalog, um eine gesellschaftlich förderliche Gestaltung und Überprüfung von

algorithmischen Systemen zu ermöglichen, die ebenso auf KI-basierte Systeme anzuwenden sind. Dazu zählen Regeln zum Kompetenzaufbau, zu wohldefinierten Verantwortlichkeiten, zur Dokumentation von Systemwirkungen und ihre Absicherung, zu Gewährleistung der Systemsicherheit und des Datenschutzes, zu Kennzeichnungspflichten, zur Sicherstellung der Nachvollziehbarkeit und Beherrschbarkeit sowie zur Gewährung von Einspruchsmöglichkeiten.

Und wie die Suche nach angemessenen Modellen der Software-Qualität und dazu passenden praktikablen Technologien der konstruktiven und analytischen Qualitätssicherung weiter geht (Singh & Kannojia, 2013), werden ebenso spezialisierte Ansätzen für KI-basierte Systeme gesucht (Russell, Dewey, & Tegmark, 2015; Seshia, Sadigh, & Sastry, 2016).

Absicherung von KI-basierten Systemen

So benötigen KI-basierte Systeme zusätzliche Methoden und Werkzeuge zur Verifikation und Validierung[8] (Van Wesel & Goodloe, 2017). Da das systematische (dynamische) Testen von Software nach wie vor die effektivste und am meisten genutzte Absicherungsmethode von Software-basierten Systemen ist (Ammann & Offutt, 2016), wird sie höchstwahrscheinlich auch eine wesentliche Methodik für die Absicherung von KI-basierten System sein. Herausforderungen an die Weiterentwicklung von Testmethoden stellen sich hierzu bezüglich Dynamik von KI-basierten Systemen (beispielweise bei Lernverfahren), die schiere Größe der genutzten Daten (um präzise Ergebnisse liefern zu können) und das Orakelproblem (um zu erwartende korrekte Ergebnisse herzuleiten) (Xie et al., 2011).

In den letzten Jahrzehnten hat die Forschung industrietaugliche Techniken zur Steigerung der Qualität, Effizienz und Zuverlässigkeit von dynamischer Tests entwickelt. Dazu gehören Ansätze wie die Automatisierung von Testausführungen mit Testtechnologien wie TTCN-3 (Testing and Test Control Notation (Grabowski et al., 2003)), für modellbasiertes Testen zur Automatisierung der Testgenerierung (MBT (Utting, Pretschner, & Legeard, 2012)) sowie der Einsatz von Such- und Optimierungsalgorithmen zur automatisierten Testauswahl und Testmengenreduktion (Harman, Jia, & Zhang, 2015). Darüber hinaus ermöglicht die Kombi-

8 Für die Validierung wird ein objektivierter Nachweis gesucht, dass Anforderungen für einen beabsichtigten Gebrauch oder eine beabsichtigte Anwendung erfüllt sind. Bei der Verifikation geht es um objektive Nachweise, dass festgelegte Anforderungen erfüllt sind. Insofern geht es grob gesprochen bei der Validierung um den Nachweis des richtigen Systems und bei der Verifikation um den Nachweis einer richtigen Realisierung eines Systems.

nation dynamischer Tests mit Verifizierungsansätzen wie Quellcodeanalyse, Modellprüfung und symbolischer Ausführung Verbesserungen beim dynamischen Testen, die die Stringenz der formalen Verifikation mit der Skalierbarkeit dynamischer Tests verbindet (Godefroid & Sen, 2018). Solche Ansätze können nicht nur zur Prüfung auf funktionale Eigenschaften, sondern ebenso auf nicht-funktionale Eigenschaften wie Leistung oder Sicherheit angewendet werden (Schieferdecker, Grossmann, & Schneider, 2012). Schließlich verbessert eine enge Integration des dynamischen Testens in den Systementwicklungsprozess und das Risikomanagement die Effizienz und Transparenz des Testens (Felderer, Grossmann, & Schieferdecker, 2018). Das Potenzial des risikobasierten Testens zur Steuerung von Testprozessen auf Basis von Unsicherheiten wurde beispielsweise im Bereich kritischer Systeme in Bezug auf Sicherheit und Gefahrenabwehr aufgezeigt und sollte ebenso für KI-basierte Systeme gelten (Erdogan, Li, Runde, Seehusen, & Stølen, 2014).

Auch wenn die Forschung für Verifikations- und Validierungsmethoden insbesondere für sub-symbolische KI-basierte Systeme am Anfang steht, ist das Testen bereits integraler Bestandteil beim Anlernen der KI: Tests werden durchgeführt, um genauere Modelle für die ursprünglichen Trainingsziele zu erhalten. Beim überwachten Lernen werden Testdatensätze zur Bewertung der KI-Modelle verwendet. So kombinieren beispielsweise (Ghosh, Lincoln, Tiwari, Zhu, & Edu, 2016) Machine Learning und Modellprüfung so, dass wenn die gewünschten logischen Eigenschaften durch ein trainiertes Modell nicht erfüllt werden, das Modell (»Modellreparatur«) oder die Daten (»Datenreparatur«), aus denen das Modell erlernt wird, systematisch verändert werden. (Fulton & Platzer, 2018) schlagen vor, die formale Verifikation mit einer verifizierten Laufzeitüberwachung zu kombinieren, um ein sicheres Lernen zu gewährleisten. DeepXplore (Pei, Cao, Yang, & Jana, 2017), DLFuzz (Guo, Jiang, Zhao, Chen, & Sun, 2018) und TensorFuzz (Odena & Goodfellow, 2018) sind Werkzeuge, die u.a. Kennzahlen zur Quantifizierung der Modellabdeckung durch Daten liefern und die Testautomatisierung erleichtern. DeepTest (Tian, Pei, Jana, & Ray, 2018) ermöglicht die systematische Prüfung neuronaler Netze, eines der gebräuchlichen Modelle in KI, unter realisti-

schen Nutzungsbedingungen, insbesondere für den Einsatz im Automobilbereich.

Neben der technischen Verifikation und Validierung können Testmethoden zudem für sozio-technische Anforderungen wie beispielsweise zur Erklärung der Wirkweisen von KI-basierten Systemen genutzt werden: Testfälle mit ihren Testdaten enthalten konkrete Ausführungsszenarien für KI-basierte Systeme und die zu erwartenden Ergebnisse (die Wirkungen). Sie können aus Nutzersicht formuliert werden und sind so leichter nachvollziehbar.

Zudem können Methoden zur Testdatengenerierung für eine Optimierung und Charakterisierung der Überdeckung von Trainingsdaten genutzt werden. Damit können merkmalsbasierte Daten mit einer repräsentativen Abdeckung des betrachteten Datenraums erzeugt werden und so unter anderem die Fairness KI-basierter Systeme durch die Berücksichtigung (Abdeckung) der relevanten Kategorien und Eigenschaften verbessern (Nguyen, Dosovitskiy, Yosinski, Brox, & Clune, 2016).

Zusammenfassung

Die Fähigkeit, KI-basierte Systeme effektiv zu entwicklen, abzusichern und zu testen, wird für die Akzeptanz solcher Systeme im Allgemeinen aber auch in sicherheitskritischen Bereichen wie Transport oder Mobilität, im Gesundheitswesen oder der Industrieautomation von zentraler Bedeutung sein. So kann u.a. die Entwicklung und Bereitstellung von Testtechnologien und -werkzeugen, von Prüfszenarien, Testfällen und -daten für KI-basierte Systeme eine solide Basis für die Verifikation und Validierung der Systeme sein. Zudem können diese dazu beitragen, KI-basierte Systeme zu erklären und so nachvollziehbarer und transparenter zu machen. Zudem können Prüfszenarien genutzt werden, um die Sicherheit KI-basierter Systeme zur Laufzeit zu gewährleisten. Perspektivisch gilt es zudem die Funktionalität und Absicherung KI-basierter Syteme in ihrer Software verschmelzen zu lassen. KI-basierte Systeme sind in ihren Ausprägungen und damit in ihren Wirkungen zu dynamisch, um Absicherungsmethoden alleinig in der Entwicklungsphase und nicht im Betrieb anzuwenden.

Die Autorin dankt dem Kompetenzzentrum Öffentliche IT bei Fraunhofer FOKUS für das Symposium »(Un)ergründlich« und die sich daraus ergebenden Anknüpfungspunkte für die wissenschaftliche Arbeit. Die Arbeit wurde teilweise vom Bundesministerium für Bildung und Forschung (BMBF) unter der Nr. 16DII111 (»Deutsches Internet-Institut«, Weizenbaum-Institut für die vernetzte Gesellschaft) sowie vom Bundesministerium für Bildung und Forschung und Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit unter der Förderungsnummer 01RIO708A4 (»Wissenschaftlicher Beirat der Bundesregierung Globale Umweltveränderungen«, WBGU) gefördert. Zudem dankt die Autorin für vielfältige Diskussionen mit Reinhard Messerschmidt, Nora Wegener, Dirk Messner und Sabine Schlacke vom WBGU, mit Stefan Ullrich, Diana Serbanescu, Gunay Kazimzade und Martin Schüssler vom Weizenbaum-Institut und mit Martin Schneider und Jürgen Großmann von Fraunhofer FOKUS.

Ammann, P., & Offutt, J. (2016). *Introduction to software testing:* Cambridge University Press.

Bertelsmann-Stiftung, & iRights.Lab. (2019). *Algo.Rules: Regeln für die Gestaltung algorithmischer Systeme,* https://www.bertelsmann-stiftung.de/fileadmin/files/BSt/Publikationen/GrauePublikationen/Algo.Rules_DE.pdf

Boehm, B. W., Brown, J. R., & Lipow, M. (1976). *Quantitative evaluation of software quality.* Paper presented at the Proceedings of the 2nd international conference on Software engineering, San Francisco, California, USA.

Breu, R., Kuntzmann-Combelles, A., & Felderer, M. (2014). New Perspectives on Software Quality [Guest editors' introduction]. *IEEE software, 31(1)*, 32-38.

Buchanan, B. G. (2005). A (very) brief history of artificial intelligence. *Ai Magazine,* 26(4), 53-53.

Dowson, M. (1997). The Ariane 5 software failure. *SIGSOFT Softw. Eng. Notes, 22(2)*, 84. doi:10.1145/251880.251992

Erdogan, G., Li, Y., Runde, R. K., Seehusen, F., & Stølen, K. (2014). Approaches for the combined use of risk analysis and testing: a systematic literature review. *International Journal on Software Tools for Technology Transfer, 16(5)*, 627-642.

Felderer, M., Grossmann, J., & Schieferdecker, I. (2018). Recent Results on Classifying Risk-Based Testing Approaches. *arXiv preprint arXiv:*1801.06812.

Fitzgerald, B. (2012). Software Crisis 2.0. *Computer, 45(4)*, 89-91.

Fromhold-Eisebith, M., Grote, U., Matthies, E., Messner, D., Pittel, K., Schellnhuber, H. J., . . . Schneidewind, U. (2019). Unsere gemeinsame digitale Zukunft. In (pp. 517). Berlin: WBGU – Wissenschaftlicher Beirat der Bundesregierung Globale Umweltveränderungen.

Fulton, N., & Platzer, A. (2018). *Safe reinforcement learning via formal methods: Toward safe control through proof and learning.* Paper presented at the Thirty-Second AAAI Conference on Artificial Intelligence.

Ghosh, S., Lincoln, P., Tiwari, A., Zhu, X., & Edu, W. (2016). *Trusted machine learning for probabilistic models.* Paper presented at the ICML Workshop on Reliable Machine Learning in the Wild.

Gibbs, W. W. (1994). Software's chronic crisis. *Scientific American, 271(3)*, 86-95.

Godefroid, P., & Sen, K. (2018). Combining model checking and testing. In *Handbook of Model Checking* (pp. 613-649): Springer.

Gordieiev, O., Kharchenko, V., Fominykh, N., & Sklyar, V. (2014). *Evolution of software quality models in context of the standard ISO 25010*. Paper presented at the Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland.

Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A., & Willcock, C. (2003). An introduction to the testing and test control notation (TTCN-3). *Computer Networks, 42(3)*, 375-403.

Guo, J., Jiang, Y., Zhao, Y., Chen, Q., & Sun, J. (2018). *DLFuzz: differential fuzzing testing*

of deep learning systems. Paper presented at the Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

Harman, M., Jia, Y., & Zhang, Y. (2015). *Achievements, open problems and challenges for search based software testing.* Paper presented at the 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST).

Hyatt, L. E., & Rosenberg, L. H. (1996). *A software quality model and metrics for identifying project risks and assessing software quality.* Paper presented at the Product Assurance Symposium and Software Product Assurance Workshop.

IEEE. (1993). Standard for Software Maintenance. (Std 1219).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature,* 521(7553), 436.

Miguel, J. P., Mauricio, D., & Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. *arXiv preprint arXiv:*1412.2977.

Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., & Clune, J. (2016). *Synthesizing the preferred inputs for neurons in neural networks via deep generator networks.* Paper presented at the Advances in Neural Information Processing Systems.

Norman, D. A. (1991). Approaches to the study of intelligence. *Artificial Intelligence, 47(1-3)*, 327-346.

Odena, A., & Goodfellow, I. (2018). Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. arXiv preprint *arXiv:1807.10875*.

Pei, K., Cao, Y., Yang, J., & Jana, S. (2017). *Deepxplore: Automated whitebox testing of deep learning systems.* Paper presented at the proceedings of the 26th Symposium on Operating Systems Principles.

Randell, B. (1979). *Software engineering in 1968.* Paper presented at the Proceedings of the 4th international conference on Software engineering.

Russell, S., Dewey, D., & Tegmark, M. (2015). Research priorities for robust and beneficial artificial intelligence. *Ai Magazine, 36(4)*, 105-114.

Schieferdecker, I., Grossmann, J., & Schneider, M. (2012). Model-based security testing. *arXiv preprint arXiv:1202.6118.*

Schieferdecker, I., Grossmann, J., & Schneider, M. A. (2019). How to Safeguard AI. In A. Sudmann (Ed.), *The Democratization of AI. Net Politics in the Era Of Learning Algorithms* (Vol. AI Critique): Transcript.

Schieferdecker, I., & Ritter, T. (2019). Advanced Software Engineering. In R. Neugebauer (Ed.), *Digital Transformation* (pp. 353-369). Berlin, Heidelberg: Springer Berlin Heidelberg.

Seshia, S. A., Sadigh, D., & Sastry, S. S. (2016). Towards verified artificial intelligence. *arXiv preprint arXiv:1606.08514.*

Singh, B., & Kannojia, S. P. (2013). *A review on software quality models.* Paper presented at the 2013 International Conference on Communication Systems and Network Technologies.

Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). *Deeptest: Automated testing of deep-neural-*

network-driven autonomous cars. Paper presented at the Proceedings of the 40th international conference on software engineering.

Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability, 22(5)*, 297-312.

Van Wesel, P., & Goodloe, A. E. (2017). Challenges in the verification of reinforcement learning algorithms.

Wirth, N. (2008). A brief history of software engineering. *IEEE Annals of the History of Computing, 30(3)*, 32-39.

Xie, X., Ho, J. W., Murphy, C., Kaiser, G., Xu, B., & Chen, T. Y. (2011). Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software, 84(4)*, 544-558.