Downloaded from Blogs on 23.11.2026 21:04h

Öffentlicher Sektor - Zukunft gestalten Blog

By PwC Deutschland | 06. Oktober 2026

Blick in den Maschinenraum der Entwicklung von SPARK Workflow – wie sich Software-Entwicklung verändert

Wie sieht der Arbeitsalltag einer Softwareentwicklerin in einem Jahr aus? Vor zwei Jahren war das eine leicht zu beantwortende Frage.

Downloaded from Blogs on 23.11.2026 21:04h

Vielleicht eine neue Version der verwendeten Programmiersprache, ein paar neue Standards, sonst das Übliche. Heute lässt sich die Frage nicht mehr so einfach beantworten. Denn wie Software entsteht, hat sich in zwei Jahren grundlegend verändert.

Spätestens seit der Veröffentlichung leistungsfähiger Frontier-Modelle wie Claude Opus 4.5 im November 2025 prägt deren Verfügbarkeit den Arbeitsalltag in der Softwareentwicklung. Die Entwicklung der KI-Lösung SPARK Workflow fiel mitten in diesen Paradigmenwechsel: Das Projekt hatte zum Ziel, eine innovative KI-Lösung zur Beschleunigung von Planungs- und Genehmigungsverfahren in nur wenigen Monaten bereitzustellen. Den Rahmen bildete eine Innovationspartnerschaft mit dem Bundesministerium für Digitales und Staatsmodernisierung (BMDS). Nach acht Monaten Entwicklung wurde die vollständige Lösung im Juni 2026 als Open Source veröffentlicht.

Neben dem Quellcode haben wir eine „**Declaration on the Use of AI Coding Assistants**“ veröffentlicht – ebenso die im Projekt entwickelten Coding-Skills, also die konkreten Methoden für die KI-gestützte Entwicklung. Damit ist SPARK Workflow möglicherweise das erste Projekt im öffentlichen Sektor, das sich offen zum KI-Einsatz bekennt und seine Regeln dazu transparent macht. Seitdem pflegen wir das Repository auf der Plattform openCode im Auftrag des BMDS öffentlich weiter. Nach drei Monaten, drei Dutzend bearbeiteten Issues und über 100 Commits ist es Zeit für einen Blick in den Maschinenraum.

Wie KI-gestütztes Entwickeln funktioniert

Große Frontier-Modelle sind auf dem öffentlich verfügbaren Weltwissen trainiert. Das umfasst Rezeptbücher, Facebook-Posts, historische Bücher und Enzyklopädien. Dazu gehören aber auch die rund 395 Millionen öffentlichen Software-Verzeichnisse auf GitHub mit mehreren Milliarden Zeilen Code sowie unzählige Foreneinträge und Diskussionen zu konkreten Code-Umsetzungen, etwa auf Stack Overflow. Dieses Wissen liegt in einer für das KI-Modell-Training günstigen Form vor. Es ist strukturiert, versioniert und oft mit einer Dokumentation dessen versehen, was die Umsetzung erreichen soll. KI-Modelle arbeiten wahrscheinlichkeitsbasiert. Diese Wahrscheinlichkeiten lassen sich auf einer großen, sauberen Datenbasis deutlich zuverlässiger schätzen. Daher sind KI-Modelle beim Verstehen und Generieren von Code so leistungsfähig.

Die nächste Abstraktionsstufe

KI-gestützte Entwicklung könnte daher die nächste Abstraktionsstufe in der Software-Entwicklung sein. In den 1940er und 1950er Jahren schrieben Entwickler:innen ihre Anweisungen als binäre Zahlenfolgen („0“ oder „1“) oder in Assembly, also direkt in der Sprache eines bestimmten Prozessors. Wer zwei Werte addieren wollte, musste wissen, in welchem Register sie liegen und wie man dorthin gelangt. Mit C kam Anfang der 1970er Jahre eine Programmiersprache, die von der konkreten Maschine abstrahierte. C++ (zunächst „C with Classes“) brachte ab 1979 die Objektorientierung, mit Klassen und Kapselung als zentralen Bausteinen. Wer in C eine Liste verwendete, musste wissen, wie sie intern aufgebaut ist, und diesen Aufbau an jeder Stelle im Programm richtig behandeln. In C++ liegen die Daten hinter der Klasse und sind nur über festgelegte Methoden erreichbar. Ein Fehler kann dadurch nur noch an einer Stelle

Downloaded from Blogs on 23.11.2026 21:04h

entstehen statt an jeder Aufrufstelle. Java brachte 1995 die nächste Schicht: Der Code läuft nicht mehr auf einem bestimmten Betriebssystem, sondern in einer virtuellen Umgebung. Die Web-Anwendung war geboren. Mit jeder Abstraktionsstufe entfernte sich der Code weiter vom konkreten Substrat und rückte näher an die Beschreibung dessen, was passieren soll. KI-gestütztes Spec-Driven Development (SDD) ist der nächste Schritt in dieser Reihe. Beschrieben wird hier in natürlicher Sprache, was die Software leisten soll – nicht in den formalen Konstrukten einer deklarativen Sprache. Diese Beschreibung ist dann das Artefakt, das gepflegt wird und auf dessen Grundlage der mit KI generierte Code entsteht.

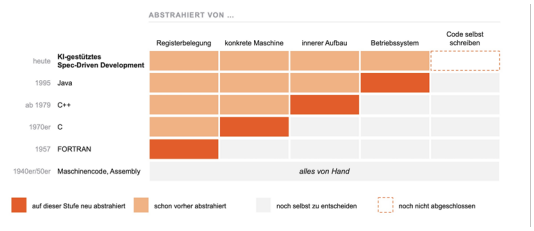


Abbildung 1: Was jede Programmiersprache zusätzlich abstrahiert

Allen diesen Sprüngen gemeinsam sind die Warnungen vor Kontrollverlust und Qualitätsverfall. Bereits 1957, gut ein Jahrzehnt vor C, entwickelten John Backus und sein Team bei IBM mit FORTRAN die erste Sprache, die Maschinencode automatisiert erzeugte, statt ihn von Hand zu schreiben. Ein großer Teil der Fachwelt hielt das Vorhaben für aussichtslos. Bei FORTRAN entstand der Maschinencode durch eine Übersetzungsmaschine – den sogenannten Compiler. Das gängigste Argument dagegen lautete, kompilierter Code könne niemals so effizient sein wie handgeschriebener Assembly-Code. Zur Kompilierung benötigte Rechenzeit war teuer, und die Vorgängersysteme lieferten Programme, die fünf- bis zehnmal langsamer liefen als handgeschriebener Assembly-Code.

Backus‘ Team baute den ersten optimierenden Compiler, dessen Ergebnisse nahe an handgeschriebenen Code herankamen. Ein Jahr später wurde bereits über die Hälfte des auf IBM-Rechnern laufenden Codes von FORTRAN erzeugt. Und über die Belegung von Prozessorregistern macht sich heute ebenso wenig jemand Gedanken wie über die Speicherverwaltung. Backus beschrieb die damalige Programmierkultur später als eine Art „Priesterschaft“, die ihr Wissen für zu komplex hielt, um es gewöhnlichen Sterblichen zuzumuten.

„… so many programmers of the freewheeling 1950s began to regard themselves as members of a priesthood guarding skills and mysteries far too complex for ordinary mortals.“

Die Analogie hat natürliche Grenzen. Ein Compiler übersetzt nach festgelegten Sprachregeln. Ein KI-Modell muss aus einer natürlichsprachlichen Beschreibung zunächst eine mögliche Umsetzung ableiten. Dabei können Anforderungen missverstanden oder unausgesprochene Annahmen ergänzt werden. Deshalb müssen Spezifikation, erzeugter Code und beobachtetes Verhalten gemeinsam geprüft werden.

Downloaded from Blogs on 23.11.2026 21:04h

Andererseits haben auch Menschen ihren Code in höheren Programmiersprachen nie rein regelbasiert geschrieben, sondern heuristisch – auf Basis von Erfahrung, Mustern und ‚Faustregeln'. Stark vereinfacht arbeiten also auch sie wahrscheinlichkeitsbasiert, mit Zugriff auf umfangreiches Wissen und erlernte Kompetenzen.

Kontextwissen ist die eigentliche Arbeit

Wer KI in der Entwicklung produktiv nutzen will, muss präzise beschreiben können, was entstehen soll. Was soll eine Funktion leisten? Mit welchen Eingaben muss sie rechnen? Wie verhält sie sich in Grenzfällen? Diese Spezifikation kommt vom Menschen, und sie muss so genau sein, dass das KI-Modell daraus ein brauchbares Ergebnis ableiten kann.

Wer zum ersten Mal mit KI-Modellen in der Entwicklung arbeitet, bemerkt vor allem, wie viele Annahmen wir normalerweise unbewusst treffen. Kolleg:innen, die das Projekt und seine Bedingungen kennen, muss man nicht alles neu erklären – sie treffen (bestenfalls) dieselben Annahmen. Deshalb fällt oft gar nicht auf, dass diese nie ausgesprochen wurden. Dem KI-Modell fehlen genau diese Informationen.

Bei der Entwicklung von SPARK Workflow haben wir das früh gemerkt. Die Lösung unterstützt bei komplexen Planungs- und Genehmigungsverfahren und muss fachrechtlich bis ins letzte Detail stimmig aufgebaut sein. Dies verlangt viel Fachwissen und Kontext, bevor überhaupt eine Zeile Code geschrieben wird. Die Entwickler:innen im Team haben sich dafür eigene Wissensdatenbanken und Glossare aufgebaut. In intensiven „Listening-Sessions" hat uns das Planungsrecht-Team von PwC Legal durch die einzelnen Prüfschritte eines Planfeststellungsverfahrens geführt. Die Termine wurden transkribiert und für KI-Modelle zugänglich gemacht.

Die Glossare brauchten wir vor allem, um sprachlich trennscharf zu bleiben. So neigte die KI beispielsweise dazu, vom Anwendungsfall Planfeststellung in den benachbarten, fachrechtlich aber eigenständigen Bereich der Bauleitplanung abzudriften. Klare Begriffsdefinitionen und ein fachlich sauber gesetzter Rahmen waren deshalb unerlässlich.

Damit verschiebt sich auch, wo die eigentliche Arbeit anfällt. Wenn die Beschreibung das gepflegte Artefakt ist, entscheidet fachliche Genauigkeit über die Qualität des Ergebnisses. Das wirkt in zwei Richtungen. Entwickler:innen brauchen mehr Wissen über die Domäne, in der sie arbeiten, und sie brauchen dieses Wissen früher. Umgekehrt rückt die Fachseite näher an die Software: Eine Spezifikation in normaler Sprache kann unser Legal-Team lesen und kommentieren, einen Quelltext nicht. Am weitesten sind wir dort gekommen, wo Fachrollen fest im Entwicklungsteam saßen und nicht nur über einzelne Termine oder Zuarbeiten beteiligt waren.

Ein verbreiteter Irrtum ist zudem, dass KI-Modelle in der Arbeit lernen. Bei ChatGPT oder Claude entsteht dieser Eindruck dadurch, dass sie sich beispielsweise an vergangene Präferenzen erinnern. Tatsächlich lernen die KI-Modelle nicht. Stattdessen speichern ChatGPT und Claude relevante Eingaben in einer eigenen Wissensdatenbank. Bei allen späteren Konversationen geben sie diese Informationen mit der Systemeingabe an das KI-Modell. Das Prinzip ist das Gleiche wie bei unseren Glossaren.

Downloaded from Blogs on 23.11.2026 21:04h

Von Wissen zu Kompetenz

So wie sich Kontextwissen über Glossare und Wissensdatenbanken aufbereiten lässt, lässt sich auch die Methodik strukturieren, nach der ein KI-Modell vorgeht. In sogenannten Skills wird festgelegt, wie das KI-Modell arbeiten soll. Relevantes Kontextwissen lässt sich direkt in den Skill integrieren. Psychologisch gesprochen ist das der Sprung vom Wissen zur Kompetenz.

In Skills fließt die praktische Erfahrung der Entwickler:innen ein. Daneben gibt es inzwischen viel Forschung dazu, welche Methoden unter welchen Bedingungen die besten Ergebnisse liefern. Rund um die besten Skills ist inzwischen ein reger Wettbewerb zwischen den Power-Usern entstanden. Ein Skill kann eine gespeicherte Instruktion sein, die immer wieder verwendet wird. Er kann aber auch Programmcode enthalten, etwa ein Python-Skript, das in bestimmten Situationen automatisch ausgeführt wird. Einige Skills nehmen die konzeptionelle Komplexität eines eigenen Programms an.

Auch im SPARK-Workflow-Team haben wir eigene Skills entwickelt, etwa für die **Qualitätssicherung** im Code oder für **Regressions-Analysen**. Es entstand bereits zu Anfang ein reger Austausch. Einige Kolleg:innen arbeiteten sich intensiv ein, entwickelten fortgeschrittene Skills und bauten tiefe Expertise auf. Von diesen ‚Early Adopters' diffundierten die neuen Fähigkeiten dann in das übrige Team.

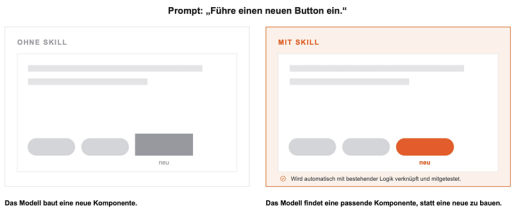


Abbildung 2: Der Spezifizieren-Erzeugen-Prüfen-Zyklus

Ein Beispiel aus dem Projekt

Wie Modelleingabe (Prompt) und Skill zusammenwirken, zeigt ein Beispiel aus der Frontend-Entwicklung. Der Prompt enthält die Anweisung, einen neuen Button einzuführen. Der Skill liefert das Kontextwissen zur Code-Basis und die Methode dazu: Das Modell soll bestehende Komponenten wiederverwenden, damit das Design konsistent bleibt. Das Modell findet daraufhin eine passende Komponente, statt eine neue zu bauen. Die Anweisungen aus dem Prompt bestimmen Platzierung und Funktion hinter dem Button. Weitere Methoden aus dem Skill legen fest, welche Tests zur Qualitätssicherung im Anschluss automatisiert stattfinden.

Downloaded from Blogs on 23.11.2026 21:04h

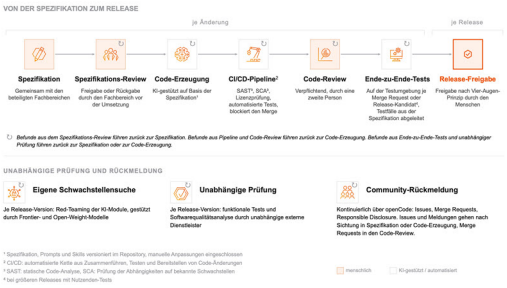
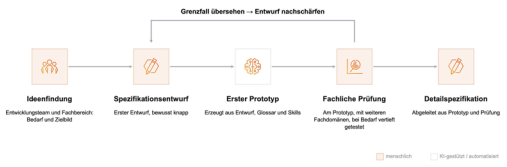


Abbildung 3: Prompt und Skill bei der Entwicklung von SPARK Workflow

Was bedeutet das für Sicherheit und Governance?

Die meisten KI-Modelle sind neben dem Generieren von Code auch in seiner Analyse stark. Das gilt auch für die Suche nach Schwachstellen. Wir mussten davon ausgehen, dass alle verfügbaren Modelle eingesetzt werden, um Schwachstellen in SPARK Workflow zu finden. Also haben wir selbst getestet und analysiert – mit einer ganzen Reihe von Frontier- und Open-Weight-Modellen. Seit der Veröffentlichung auf openCode setzen wir diese Analysen fort und weisen auf gefundene Schwachstellen über Issues hin. Beim **Hackathon** des BMDS standen virtuelle Workstations mit Zugang zu leistungsstarken Frontier-Modellen bereit, um die Sicherheit der Lösung in einem „Red-Teaming“ zu überprüfen. Diese praktischen Analysen ergänzen die klassische Codeanalyse (Static Application Security Testing, SAST) und die Prüfung der Abhängigkeiten (Software Composition Analysis, SCA), die weiterhin unverzichtbar bleiben.

Für den Umgang mit KI-generiertem Code gelten zudem mindestens dieselben Standards wie für handgeschriebenen Code. Er wird gesichtet, getestet und unabhängig freigegeben – von der Freigabe der Spezifikation durch den Fachbereich über das verpflichtende Code-Review durch eine zweite Person bis zur Release-Freigabe im Vier-Augen-Prinzip durch den Menschen. So wie Code aus einem Online-Forum nicht unbesehen übernommen werden sollte, sollte auch KI-generierter Code immer kritisch überprüft werden. Diese Vorgaben haben wir uns im Projekt selbst gesetzt und in der bereits eingangs erwähnten „**Declaration on the Use of AI Coding Assistants**“ verschriftlicht.



Downloaded from Blogs on 23.11.2026 21:04h

Abbildung 4: Wie KI-generierter Code bei SPARK Workflow abgesichert und freigegeben wird

In Abstimmung mit dem BMDS haben wir außerdem die entwickelten Skills vollständig als Open Source veröffentlicht. Damit lässt sich nachvollziehen, wie bestimmte Ergebnisse entstanden sind. Und wer die Lösung unabhängig von uns weiterentwickelt, erhält relevantes Kontextwissen und „Kompetenzen“.

Unsere Lessons Learned

Allein seit November 2025 gab es mehrere größere Sprünge bei der Leistungsfähigkeit von KI-Modellen. So wie wir die verwendeten Skills veröffentlicht haben, teilen wir auch unsere Erfahrungen aus den letzten Monaten. Wir hoffen auf eine Diskussion darüber, wie sich KI-gestützte Entwicklungsmethoden im öffentlichen Sektor einsetzen lassen.

1. Austausch und Neugier im Team: In größeren Teams gehen nie alle im gleichen Tempo. Einige arbeiten sich früh in ein neues Werkzeug ein, andere erst Monate später. Dazu kommt, dass neue Methoden meist zuerst in einem Themengebiet auftauchen. Wer sich mit IT-Sicherheit befasst, stößt auf andere Werkzeuge als jemand, der an der Benutzeroberfläche arbeitet. In beiden Fällen liegt neues Wissen zunächst nur an einer Stelle im Team. Der Wissenstransfer bringt es in die Breite. Bei uns lief das über verpflichtende gegenseitige Code-Reviews, viele Austauschformate im Team und aktive Chat-Diskussionen, in denen neue Werkzeuge bewusst zum Thema gemacht wurden. Die Projektleitungen tragen eine besondere Verantwortung. Wer vorlebt, dass Ausprobieren erwünscht ist und Scheitern zur Lernkurve gehört, schafft die Kultur, die eine Veränderung dieser Größenordnung braucht.

2. Ein gemeinsames Repository für Skills: Das ist die technische Grundlage für den Wissenstransfer und dafür, dass Skills kollaborativ weiterentwickelt werden. Kontextwissen und Skills gehören genauso in jedes Repository wie Readme oder Lizenzbestimmungen. Wir haben von Anfang an ein internes Repository gepflegt, in dem alle ihre eigenen Skills in Branches ablegen konnten. Besonders gelungene Skills haben wir in den Main-Branch und schließlich in das öffentliche openCode-Repository übernommen.

3. Laufende Begleitung durch IT-Sicherheit und Datenschutz: Ein eigenes Team hat uns durchgehend begleitet und war dadurch fachlich tief eingebunden. Das hatte drei Effekte: Erstens konnten wir wichtige Entscheidungen schnell und mit der nötigen Sorgfalt treffen. Zweitens sank durch die feste Einbindung ins Team und ein Gefühl von gemeinsamer „Mission“ die Hemmschwelle für den Austausch zu kritischen Themen. Risiken wurden dadurch früher und mit mehr Offenheit besprochen. Drittens konnten unsere IT-Sicherheits- und Datenschutz-Expert:innen die neuen Entwicklungsmethoden selbst im Projektalltag erleben, damit Erfahrungen sammeln und deren Risiken laufend und praxisnah bewerten.

4. Automatisierte Tests werden wichtiger: Je mehr KI-gestützt entwickelt wird, desto relevanter werden neben den Peer-Reviews die deterministischen Prüfungen in der CI/CD-Pipeline. Wir haben früh eine eigene Pipeline aufgebaut, die neben Sicherheitsprüfungen (SAST, SCA) auch Softwarelizenzen gegen vorgegebene Open-Source-Allowlists abgleicht. Ebenso wichtig sind umfangreiche Test-Suiten. Automatisierte Integrations-, Regressions- und Ende-zu-Ende-Tests ermöglichen es, die Robustheit des Codes nach größeren KI-gestützten Änderungen schnell zu bewerten.

Downloaded from Blogs on 23.11.2026 21:04h

5. Der Blick von außen: Bereits im Frühjahr 2026, mehrere Wochen vor der ersten Open-Source-Veröffentlichung, begannen unabhängige externe Dienstleister mit umfangreichen funktionalen Tests und Analysen der Softwarequalität (SQA). Dieser Prozess kann schmerzhaft sein. Sein Wert liegt aber genau darin, dass die Prüfenden nicht zum Projekt gehören. Sie teilen unsere Annahmen nicht, stellen Fragen, die im Team niemand mehr stellt, und stoßen auf Probleme, die wir uns selbst gar nicht mehr vorstellen konnten. Diese Perspektive lässt sich in keinen Skill schreiben.

Wie sieht nun also der Arbeitsalltag einer Softwareentwicklerin in einem Jahr aus? Wir wissen es nicht sicher, aber die Indizien deuten darauf hin, dass KI-gestütztes Spec-Driven Development bleiben wird. Statt die Funktion selbst zu schreiben, hält die Entwicklerin mit einem interdisziplinären Team fest, was sie leisten soll, liest den erzeugten Code gegen, bewertet die Testergebnisse und schärft nach …

Autorinnen und Autoren waren Dominik Lawetzky, Gernot Bahle, Lauren Garcia Bernerund Paulina Kuhndörfer sowie mitgewirkt haben Dr. Nicolai Bieber, Dr. Tobias Franke und Hannes Schroter.

Ansprechpartner:
Dominik Lawetzky

Zu weiteren PwC Blogs

Schlagwörter

Artificial Intelligence (AI), Datenschutz, Digitalisierung, IT-Sicherheit, IT-Systeme, Informationstechnologie (IT), Software

Kontakt



Prof. Dr. Rainer Bernnat
Frankfurt am Main
rainer.bernnat@pwc.com