Das Ende einer Ära: Vom Programmierhandwerker zum KI-Regisseur[1]

Dietrich Boles, 13.08.2026

Wenn ich heute vor meinem Bildschirm sitze, denke ich manchmal an den Moment zurück, als alles begann. Es war Anfang der 1980er Jahre. Ich war etwa 17 Jahre alt und hatte mir von meinem mühsam Ersparten einen programmierbaren Taschenrechner gekauft – den HP-41C. Ich hatte nie gelernt zu programmieren; es gab niemanden, der es mir zeigte, und die Materie war mir vollkommen fremd. Register, Adressen und Speicherwerte erschienen mir im ersten Moment tief suspekt. Und doch lag darin eine ungeheure Magie: Ich fügte mir alles im Selbststudium zusammen, tüftelte Tag für Tag an der Logik und versuchte zu verstehen, wie ich dieser kleinen Maschine etwas beibringen konnte, damit sie Probleme für mich löste. Als es mir schließlich durch pures Ausprobieren und Durchbeißen gelang, das Spiel *Superhirn* darauf zu programmieren – bei dem der Rechner eine n-stellige Zufallszahl generierte, mich zur Eingabe einer entsprechenden Zahl aufforderte und mir Rückmeldung über Anzahl und Position richtiger Ziffern gab –, war es um mich geschehen. Es war eines dieser Erlebnisse, die mich dazu bewogen haben, mein gesamtes berufliches Leben in diese Richtung lenken.

Nach meinem Abitur folgte ab 1983 die Ausbildung zum Mathematisch-Technischen Assistenten an einem Großforschungsinstitut nahe Bonn. Ich lernte Sprachen wie Elan, Modula-2, Cobol und Fortran, arbeitete mich selbstständig in Lisp ein und verfiel der Faszination von Prolog. Nach der Ausbildung wechselte ich1986 an diesem Institut in ein Projekt, in dem ich die parallele Programmierung kennenlernte – eine ganz neue Welt voll von Nebenläufigkeiten, Synchronisationsproblemen und tückischen Race Conditions, die sich oft erst beim zehntausendsten Durchlauf zeigten.

Diese Komplexität begeisterte mich und forderte mich so sehr heraus, dass ich 1988 nach Oldenburg wechselte und dort noch Informatik studierte und 2002 dort auch promovierte.

Ende der 1980er Jahre entstanden die ersten wirklichen objektorientierten Ideen. Ich lernte C++ und programmierte viele Jahre darin, ehe das Web aufkam und ich von C++ zu Java wechselte. Die Objektorientierung war eine völlig andere Welt als die imperative Programmierung: Man dachte nicht mehr nur in der zeitlichen Zerlegung von Abläufen, sondern in der strukturellen Zerlegung von Problemen und Gedanken. Wir arbeiteten mit Frameworks, entwickelten eigene Frameworks, wendeten intuitiv Design Pattern an oder entdeckten und etablierten eigene Muster.

Mit der Entwicklung von Webanwendungen ab Mitte der 1990er Jahre erschloss sich wieder eine ganz neue Kategorie. Man brauchte ein Frontend mit HTML, CSS und JavaScript und ein solides Backend, für das ich anfangs Java nutzte und später zu PHP wechselte. Völlig neue Aspekte kamen hinzu: Security, das Aufspüren und Schließen von Sicherheitslücken, und irgendwann auch die stetig wachsenden Anforderungen des Datenschutzes.

Während meiner Promotion an der Universität Oldenburg übernahm ich in der Lehre die Programmierausbildung der Erstsemester im Informatik-Studiengang. Es lag mir

[1] Inspiriert von https://www.jamesdrandall.com/posts/the_thing_i_loved_has_changed/

vor allem daran, die Studierenden von Grund auf zu begeistern. Ich zeigte den Studierenden, wie man Programme schreiben kann, die selbstständig schachähnliche Strategiespiele spielen konnten – basierend auf Minimax-Algorithmen und Alpha-Beta-Pruning. Ich organisierte Turniere, in denen die studentischen Programme gegeneinander antraten. Es war faszinierend mitanzusehen, wie diese Algorithmen glanzvoll siegten – oder manchmal kläglich scheiterten.

Ich entwickelte eigene didaktische Werkzeuge für die Programmierausbildung wie den Hamster-Simulator, experimentierte mit visuellen Block-basierten Programmiersprachen. Mit meinen Studierenden trat ich bei echten Programmierwettbewerben an. Wir nahmen bspw. am NWERC teil, dem europäischen Programmierwettbewerb für Studierende, und gewannen ihn 2001 sogar. Wir reisten 2002 zu den ICPC World Finals nach Hawaii – den studentischen Programmier-Weltmeisterschaften – und belegten dort unter 64 internationalen Spitzenmannschaften einen hervorragenden 18. Platz.

Über fast vier Jahrzehnte hinweg war das Programmieren für mich niemals bloß ein Beruf, sondern eine echte Berufung und gelebte Handwerkskunst. Es war der Reiz, tagelang hartnäckig nach einem verdeckten Fehler zu suchen, komplexe Software-Architekturen, Schichtenmodelle und Design Pattern tief zu durchdringen – und schließlich dieses unbeschreibliche Aha-Erlebnis zu spüren, wenn der Code nicht nur funktionierte, sondern höchsten Qualitätsansprüchen genügte. Das meisterhafte Lösen von Problemen durch eigene Geistesarbeit, das Erschaffen exzellenter Software und das Entfachen genau dieser Leidenschaft bei meinen Studierenden bildeten das fundamentale Herzstück meiner beruflichen Identität.

Und dann kam der November 2022.

Ich hatte von ChatGPT gehört, öffnete neugierig die Oberfläche und fütterte das System testweise mit den Aufgaben meiner letzten Prüfungsklausur. Die Anforderungen waren anspruchsvoll: *„Gegeben ist folgendes Problem – schreiben Sie ein Java-Programm, das es löst.“* Keine Minute später stand das Ergebnis auf dem Bildschirm. Fehlerfrei, volle Punktzahl. Völlig ungläubig nahm ich eine zweite Klausur – und wieder lieferte die KI 100 Prozent. Ich saß da, gelähmt von einem tiefen Schock und zugleich ergriffen von einer immensen Faszination. Niemals hätte ich es für möglich gehalten, dass eine Maschine eigenständig und in Sekundenschnelle solch komplexe Gedankengänge vollziehen kann. In diesem Augenblick geriet mein gesamtes Fundament aus vier Jahrzehnten Programmierung und Lehre ins Wanken: Wozu bildeten wir überhaupt noch Menschen in Programmierung aus, wenn man statt eigener Denkarbeit nur noch Prompts formulieren musste?

2025 hat OpenAI übrigens ein KI-System bei den ICPC World Finals gegen die Wettbewerbsaufgaben antreten lassen. Ich weiß aus eigener Erfahrung, wie extrem schwer diese Aufgaben sind – wir hatten damals von zwölf Aufgaben vier gelöst, während die besten Teams sechs schafften. Die KI hingegen löste alle zwölf Aufgaben. Wäre sie unter den regulären Wettbewerbsbedingungen gewertet worden, hätte dieses Ergebnis für den ersten Platz gereicht.

Spätestens an diesem Punkt fragt man sich völlig entsetzt: Braucht man dieses tüftelnde Denken und Problemlösen heute überhaupt noch? Wie soll man Studierende

künftig noch dazu begeistern, Schachprogramme zu schreiben oder an Programmierwettbewerben teilzunehmen, wenn alle genau wissen, dass man dieses Wissen eigentlich nicht mehr braucht, weil die KI es ohnehin um Längen besser kann?

Zwar beruhigte ich mich vorübergehend mit dem Gedanken, dass KI bei größeren Systemen mit vielen Klassen noch scheiterte und wir Menschen zumindest das architektonische Denken behalten würden. Doch die Entwicklung blieb nicht stehen. Als ich vor Kurzem moderne KI-Werkzeuge, wie Antigravity von Google, für die Entwicklung einer komplexen Webanwendung einsetzte, geschah das für mich wiederum Unfassbare: Die KI erzeugte nicht nur saubere einzelne Klassen, sondern eine durchdachte Gesamtarchitektur, die in ihrer Qualität besser war als fast alles, was ich in den letzten 30 Jahren von Studierenden zu Gesicht bekommen hatte.

Heute hört man dennoch oft das Mantra: Programmierer werden zwar nicht arbeitslos, aber ihre Aufgaben verlagern sich – sie müssen keinen Code mehr selbst tippen, sondern KI-generierten Code überprüfen, umfangreiche Tests schreiben, präzise Anforderungen formulieren und komplexe Softwarearchitekturen entwickeln.

Doch dieser Behauptung kann ich so nicht mehr zustimmen; sie ist im Grunde schon wieder überholt. Moderne KIs entwerfen bereits heute hochgradig strukturierte Architekturen und schreiben meilenweit bessere Tests mit nahezu vollständiger Abdeckung. Selbst der vielzitierte Bereich der Anforderungsdefinition wandelt sich grundlegend: Man muss der KI nicht mehr jedes Detail in einer lückenlosen Spezifikation vordiktieren. Oft reicht ein grober Rahmen, und die KI antizipiert fachliche Anforderungen, denkt Sonderfälle mit und schlägt von sich aus sinnvolle Erweiterungen vor.

Auch das Argument der mangelnden Zuverlässigkeit hält der Praxis nicht mehr stand. Erst neulich hörte ich in einer Diskussion den besorgten Einwand: *„Ich würde niemals in ein Flugzeug steigen, dessen Steuerungssoftware von einer KI generiert wurde.“* Ich finde diese Haltung im Grunde absurd. Wenn ich auf meine Jahrzehnte in der Lehre zurückblicke, enthielten die Programme meiner Studierenden stets deutlich mehr Fehler als das, was aktuelle KIs auf Anhieb liefern. Warum vertrauen wir dem Menschen an dieser Stelle eigentlich so viel mehr als der Maschine?

Die Notwendigkeit, Software gründlich zu prüfen und auf Herz und Nieren zu testen, gab es schließlich schon immer – völlig unabhängig davon, wer den Code geschrieben hat. Doch diese Aufgabe ist heute sogar um ein Vielfaches einfacher und weniger zeitintensiv geworden, schlicht weil KI-generierter Code von Grund auf mit bedeutend weniger Fehlern behaftet ist.

Und was die Qualitätskontrolle selbst betrifft, erledigt eine KI das Lesen und Prüfen von Code heute nicht nur gründlicher, präziser und um ein Vielfaches schneller als der Mensch – man kann heute auch schlicht eine KI selbst einsetzen, um die Erzeugnisse einer anderen zu auditieren. Wie überlegen KI in dieser Disziplin agiert, zeigte mir neulich ein eigenes Experiment: Ich gab einer KI große studentische Projekte mit hunderten von Klassen zusammen mit meinen detaillierten Bewertungskriterien und bat sie um eine Fehleranalyse. Das Ergebnis war verblüffend: Die KI deckte Schwachstellen auf, die selbst ich als erfahrener Dozent manuell niemals entdeckt hätte.

Die Vorstellung, dass das manuelle Prüfen, Testen, Spezifizieren und Entwickeln von Architekturen die Zukunft der menschlichen Programmierarbeit dauerhaft sichern wird, greift daher viel zu kurz – das werden mit Sicherheit nicht die Hauptaufgaben sein, die uns Entwicklern bleibt.

In den letzten Wochen wurde mir immer mehr klar: Die Faszination des Selberschreibens ist zu Ende. Ich werde in meinem Leben nie wieder eine einzige Zeile Code von Hand tippen. Das nächtelange Zeilen-Tippen und Fehlersuchen hat ausgedient.

Meine Rolle hat sich grundlegend gewandelt. Ich tippe nicht mehr, ich führe Regie. Ich fungiere wie ein Regisseur, der eine Armada von Programmierern unter sich hat. Nur sind das keine Menschen, sondern KI-Agenten. Ich definiere Anforderungen mündlich, steuere Prozesse, lenke nach, nehme Korrekturen vor. Begriffe wie Schleifen, Arrays oder Klassen muss man nicht mehr in den Mund nehmen; man denkt in Anforderungen und fachlichen Zusammenhängen.

Kürzlich entwickelte ich eine umfangreiche Webanwendung. Hätte ich sie selbst programmiert, hätte mich das vermutlich ein ganzes Jahr intensiver Arbeit gekostet. Mit der Unterstützung der KI war das gesamte Projekt nach gerade einmal drei Tagen abgeschlossen – funktional, visuell und architektonisch besser, als ich es mir je erträumt hätte.

Dabei wird auch deutlich, wie klassische Software-Tugenden wie Wartbarkeit oder Wiederverwendbarkeit ihren gewohnten Sinn verlieren. Hohe Softwarequalität ist heute kein Selbstzweck mehr, um Code mühsam für die Zukunft erweiterbar zu halten. Wenn sich in drei Jahren die Rahmenbedingungen ändern oder eine neue Funktion gebraucht wird, gibt man den bestehenden Code einfach der KI mit der entsprechenden Anforderung – und sie baut ihn mühelos um, vollkommen unabhängig davon, wie gut oder schlecht die bisherige Qualität der Codebasis war.

Das ist auf der einen Seite tief befriedigend und unglaublich spannend. Ich habe in den letzten Wochen so viele Anwendungen realisiert, die ich mir schon immer gewünscht hatte, für deren Entwicklung mir aber schlicht die Zeit fehlte. Auf der anderen Seite fehlt da etwas. Die Nähe zur Materie ist weg. Das tüftelnde Puzzeln, die Jagd nach dem Fehler und das tiefe Befriedigungsgefühl, wenn es nach langem Ringen endlich „Klick“ macht – all das ist zusammengepresst worden zu einem einfachen Spiel aus Prompt und Antwort.

Wie wird das alles weitergehen, wie wird es in 2 bis 3 Jahren aussehen? Ich weiß es nicht, ich kann es mir auch nicht vorstellen. Meine Vermutung ist, dass wir einer KI bald nur noch zwei, drei Sätze hinwerfen müssen. Sie wird unsere Gedanken weiterdenken, Anforderungen antizipieren, kurz Rückfrage halten und nach wenigen Minuten ein fertiges System bereitstellen, das all unsere Wünsche mehr als erfüllt. Der Programmierer bzw. Softwareentwickler im klassischen Sinne wird dann schlicht nicht mehr gebraucht.

Das beinhaltet eine fast tragische historische Parallele: Die Industrialisierung und die Erfindung von Maschinen führten einst dazu, dass unzählige handwerkliche und landwirtschaftliche Berufe wegfielen. Jahrzehnte später waren es schließlich wir Informatiker, die mit unserer Software die Arbeit unzähliger anderer Berufsgruppen

automatisierten. Und nun erleben wir die Ironie der Geschichte: Wir Informatiker haben uns im Grunde selbst rationalisiert, zumindest was die Programmierung bzw. Softwareentwicklung angeht. Durch die Erschaffung der KI haben wir das Handwerk, das uns einst definierte, schrittweise überflüssig gemacht.

Ich gehe Ende dieses Jahres in Rente und stehe nun an einem entscheidenden Wendepunkt. Es ist ein bewusstes Innehalten in einer Zeit des fundamentalen Wandels. Die Identität, die ich über vier Jahrzehnte hinweg als Entwickler und Lehrender aufgebaut habe, passt nicht mehr zur neuen Realität. Das Handwerk, das ich so geliebt habe, hat sich aufgelöst. Nun müssen wir uns ganz neuen Fragen stellen: Wie sieht die Arbeitswelt für diejenigen aus, die heute noch als Programmierer arbeiten beziehungsweise zum Programmierer ausgebildet werden? Welche Kompetenzen brauchen sie wirklich? Und wie muss eine universitäre Ausbildung aussehen, wenn das klassische Programmieren eigentlich nicht mehr gelernt werden muss?

Die Magie von damals, als ich dem HP-41C erstmals Leben einhauchte, ist nicht gänzlich verschwunden – aber sie hat ihre Gestalt für immer dramatisch verändert.

„Ich habe fertig!“