KI-gestütztes Entwickeln mit IntelliJ IDEA, Visual Studio IntelliCode und Tabnine

KInematik

Bernhard Steppan

KI-Systeme zur Textkomplettierung sind alltäglich. Ob sie auch als Programmierhilfe überzeugen, testen wir am Beispiel zweier IDEs und der KI-Erweiterung Tabnine.

JetBrains IntelliJ IDEA und Microsofts Visual Studio Code bieten von Haus aus gute Programmierhilfen. IntelliJ IDEA enthält bereits eine durch maschinelles Lernen gestützte Codevervollständigung, während Visual Studio Code sich mit dem KI-Tool IntelliCode wappnet. Darüber hinaus gibt es für beide einige externe KI-Erweiterungen wie Kite und Tabnine, die versprechen, die Produktivität von Entwicklerinnen und Entwicklern nochmals zu steigern.

Die genannten KI-Erweiterungen funktionieren wie die Textvorschläge, die man auf einem Smartphone während der Texteingabe im Editor bekommt. Man tippt einen Text ein und der Editor bietet mehrere Vorschläge an, mit denen man den Text fortsetzen könnte. In der Programmierung ist das als Autocomplete bekannt. Die normale Codevervollständigung listet jedoch nur die nächste sinnvolle Ergänzung auf, wie sie sich zum Beispiel über eine Codeinspektion ermitteln lässt. KI-Erweiterungen können das übertreffen, wenn sie mit passenden Programmbeispielen angelernt wurden. Dann sind sie in der Lage, komplette Textblöcke ohne Nacharbeit zu ergänzen, was die Programmiergeschwindigkeit erheblich steigert.

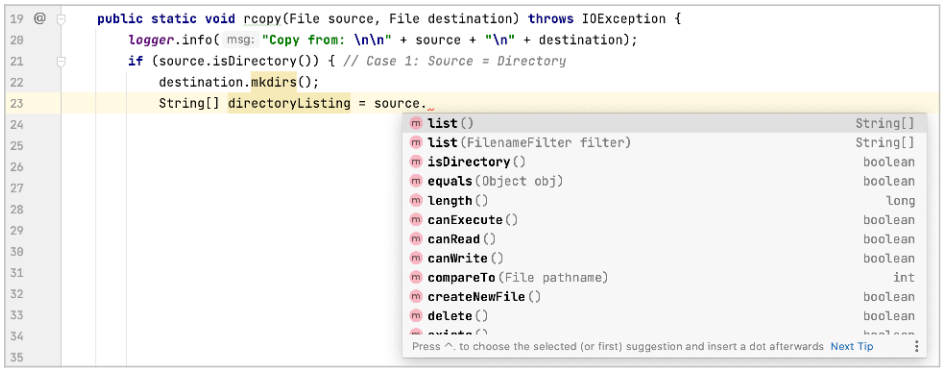
Um das zu testen, haben wir zwei unterschiedliche Testsuiten zusammengestellt. Die darin enthaltenen Tests prüfen, wie die Tools sich in sehr unterschiedlichen Aufgabenstellungen bewähren. Die erste Testsuite besteht aus einem Java-Programm zum rekursiven Kopieren von Verzeichnisbäumen und prüft, wie technisch orientierte Klassen mit Konzepten wie Konstruktoren, Getter- und Setter-Methoden und Ausnahmebehandlung ergänzt werden. Zudem untersucht sie mit Refactoring, ob die KI-Funktionen beim Extrahieren von Methoden und beim Umbenennen ins Trudeln kommen.

Die zweite Testsuite ist ein fachlich orientiertes Java-Programm zur Bestellaufnahme mit Klassen in deutscher Sprache. Die Tests sind anders ausgerichtet als die des ersten Durchlaufs: Sie versuchen herauszufinden, wie die KI-Tools auf Klassen reagieren, auf die sie mit hoher Wahrscheinlichkeit gar nicht trainiert werden konnten. Hintergrund der Tests dieser Suite ist der Umstand, dass viele Firmen für die Geschäftslogik ihrer Programme fachlich

iX-TRACT

- IDEs wie IntelliJ IDEA und Visual Studio Code enthalten Programmierhilfen wie eine Codevervollständigung, die Entwicklern die Arbeit erleichtern.
- Zusätzliche KI-Erweiterungen wie Tabnine gehen über die reguläre Codevervollständigung hinaus, indem sie nicht nur die nächste sinnvolle Ergänzung listen, sondern ganze Textabschnitte.
- Zwei zum Vergleich dieser IDEs erstellte Testsuiten ermitteln, wie die IDEs Klassen ergänzen, wie die KI-Tools sich beim Extrahieren von Methoden verhalten und auf Klassen ohne Datenbasis reagieren.

benannte Klassen- und Methodennamen verwenden, auf die man KI-Tools schlecht vorbereiten kann. Der Quellcode solcher Klassen ist meistens nicht öffentlich zugänglich. Es ist zudem nicht ungewöhnlich, diese Programme in der jeweiligen Landessprache zu schreiben. Daher sind die Klassen, Methoden und Variablen dieses Tests bis auf die Java-Schlüsselbegriffe durchgängig in deutscher Sprache verfasst. Der Test soll vor allem untersuchen, ob die KI-Erweiterungen durch solche fachlichen Klassen aus dem Tritt kommen und ob sich in dem vergleichsweise kurzen Zeitraum im Rahmen dieses Artikels überhaupt ein Lerneffekt zeigt.



Die Codevervollständigung von IntelliJ IDEA sortiert ihre Vorschläge so, dass sich die besten oben befinden. Hier liefern die beiden ersten Funktionen wie gewünscht ein String-Array (Abb. 1).

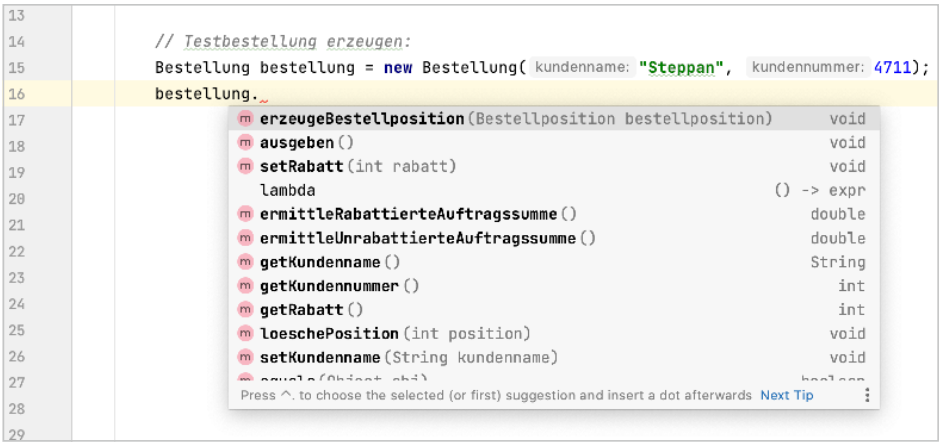
JetBrains IntelliJ IDEA ohne zusätzliche KI

Der erste Kandidat ist die integrierte Entwicklungsumgebung (IDE) IntelliJ IDEA. Sie ist seit 2001 auf dem Markt und liegt in einer kostenfreien Community Edition als Open Source und in der etwas leistungsfähigeren, aber kostenpflichtigen Ultimate Edition vor. Diese IDE ist für Windows, macOS und Linux verfügbar. Dem Test liegt die Community Edition für macOS in der neuesten Version 2021.1.2 zugrunde. Die Testprojekte sollte IntelliJ zunächst ohne eine externe KI-Erweiterung absolvieren, da die Entwicklungsumgebung bereits über umfangreiche Autovervollständigungs- und Refactoring-Funktionen verfügt. Darunter befindet sich auch die KI-Funktion namens Machine-Learning-Assisted Code Completion (siehe ix.de/zmtr).

Die erste Testsuite soll das Kopierprogramm für Verzeichnisbäume implementieren, das aus zwei Java-Klassen besteht. Die erste Klasse, `CopyProcess`, enthält den Kopieralgorithmus, die zweite Klasse, `TransferApp`, die Verarbeitung der Eingabeparameter und den Aufruf der rekursiven Kopierfunktion. Das Implementieren des Kopieralgorithmus gestaltet sich sehr einfach: Für das Anlegen des Ziel-Packages kann man entweder den Bezeichner des Pakets direkt in die neue Klasse schreiben und anschließend die Klasse per Refactoring in das korrekte Package verschieben. Oder man legt das Package im Projektmanager an und erzeugt danach gleich dort die Klasse.

Die stets sinnvollen und syntaktisch richtigen Vorschläge unterstützen das Implementieren der rekursiven Kopiermethode sehr gut. Abbildung 1 zeigt ein Beispiel für das Ermitteln eines Arrays mit Elementen des Typs `String`. Bei Kontrollstrukturen wie Schleifen reicht es aus, im Editor ein Java-Schlüsselwort wie `for` einzufügen. Die IntelliJ-Programmierhilfe bietet daraufhin die zwei möglichen For-Schleifenarten von Java zum Generieren an.

`Try-Catch`-Blöcke und `Throws`-Statements ergänzt IntelliJ nicht einfach während des Schreibens, was meistens auch nicht sinnvoll ist. Um bei der Ausnahmebehandlung mit wenig Schreibarbeit zum Ziel zu kommen, ist es am besten, zunächst den Methodenaufruf zu schreiben, der eine Exception auslösen kann. Danach gilt es zu entscheiden, ob IntelliJ den erforderlichen Try-Catch-Block erzeugen oder die Exception von der aufrufenden Methode behandelt werden soll. Um aus ganzen Blöcken Code zu generieren, bietet IntelliJ viele Templates an, die sich per Shortcuts oder Kontextmenü einsetzen lassen. So implementieren Entwicklerinnen und Entwickler Konstruktoren, Getter- und Setter-Methoden und andere Codefragmente mit wenig Schreibarbeit. Wählen sie den anderen Weg und tippen Anweisungen direkt in den Editor ein, enthalten die Vorschläge zum Vervollständigen meist keine größeren Codeblöcke, sind dafür aber stets syntaktisch korrekt.



Test mit fachlichen Klassen: Die Programmierhilfe zeigt gleich an erster Stelle den inhaltlich passenden Vorschlag mit dem korrekten Parameter als Ergänzung an (Abb. 2).

Erneut auf den Zahn gefühlt

Das Implementieren eines fiktiven ERP-Programms, das beliebige Artikel bestellt, bildet den zweiten Testlauf. Das Programm besteht aus den drei fachlichen Klassen `Artikel`, `Bestellposition` und `Bestellung`, die in deutscher Sprache vorliegen. Das Datenmodell setzt sich aus Artikeln zusammen, die sich zu Bestellpositionen zusammenfassen lassen. Mehrere dieser Positionen ergeben wiederum eine Gesamtbestellung.

Der Test zeigt keinen nennenswerten Unterschied zum ersten Testlauf. Die fachlichen Klassen mit ihren Attributen sowie Getter- und Setter-Methoden sind ebenso zügig wie bei den technischen Testfällen implementiert. Ein Beispiel dafür ist das Erzeugen einer Bestellposition mit der Methode `erzeugeBestellposition()` der Klasse `Bestellung` (Abbildung 2).

Als Ergebnis der Testläufe zeigt sich, dass sich die meisten Teile der Java-Testklassen mit einem Minimum an Schreibar-

beit implementieren lassen. Die Vorschläge waren hierbei in den meisten Fällen passend und zudem syntaktisch korrekt. Die Erwartung, dass ganze Codeblöcke während des Schreibens von der Codevervollständigung ergänzt werden, konnte IntelliJ mit „Bordmitteln“ nur durch die Codegenerierungsfunktionen auf Basis seiner Templates erfüllen. Ein Lerneffekt bei der Implementierung, wie er bei anderen KI-Autocomplete-Tools der Fall ist, konnte bei IntelliJ nicht beobachtet werden. Trotzdem ist die Programmierhilfe von IntelliJ IDEA überdurchschnittlich hilfreich.

KI an Bord: IntelliJ IDEA mit Tabnine

Die Firma Codota hat Tabnine samt seinem KI-Tool Ende 2019 übernommen und sich danach in Tabnine umbenannt. Daher gibt es sowohl Tabnine- als auch Codota-Plug-ins für Entwicklungstools (siehe Kasten „KI-Plug-ins für Eclipse IDE“). Das KI-Plug-in Tabnine basiert auf der Text-KI GPT-2 von OpenAI – einem Unternehmen, das unter anderem Microsoft und Elon Musk finanzieren. Es hat sich auf das Erforschen der künstlichen Intelligenz spezialisiert. GPT-2 steht für Generative Pre-trained Transformer 2 und basiert auf einer Transformer-Netzwerkarchitektur.

Zusätzlich zum öffentlichen GPT-2-Modell verwendet Tabnine ein privates lokales Modell, in das die Trainingsergebnisse während des Programmierens mit dem KI-Tool einfließen. Die Firma Tabnine garantiert hierbei ausdrücklich, dass dieses lokale Modell und der gesamte Quellcode nicht weitergegeben wird.

Das öffentliche Modell wird mit dem Quellcode von Open-Source-Programmen angelernt, deren Lizenzen eine solche Analyse mit einer kommerziellen Verwertung gestatten. Es enthält als Wissensbasis momentan ungefähr 2 Millionen Quellcodedateien. Die kostenfreie Version des KI-Tools basiert auf einem relativ einfachen Modell mit „nur“ 124 Millionen Parametern. Die kommerzielle Variante verwendet ein Modell mit 350 Millionen Parametern und ist laut Hersteller daher treffgenauer. Die Vorschläge zum Codevervollständigen sollen nicht nur die Produktivität beim Programmieren steigern, sondern auch ein Auswendiglernen der Programmsyntax überflüssig machen.

Tabnine beherrscht alle wichtigen Programmiersprachen wie JavaScript, Python, PHP, C/C++, HTML/CSS, Go, Java, Ruby, C#, SQL und Kotlin. Es gibt Plug-ins für Entwicklungswerkzeuge wie Android Studio, Atom, IntelliJ IDEA und Visual Studio Code. Für den Test haben wir die kommerzielle Version Tabnine Pro in der Engine-Version 3.5.3 als Plug-in für IntelliJ verwendet, da diese Version die bessere Vorhersagegenauigkeit besitzen soll. Man installiert sie entweder über die Website von Tabnine oder den Marktplatz von JetBrains. Zunächst sollte das technische Kopierprogramm für den ersten Testlauf implementiert werden. Hierbei wechselten sich hilfreiche Vorschläge mit solchen ab, die unbrauchbar und syntaktisch falsch waren.

KI-Plug-ins für die Eclipse IDE

Für die Eclipse IDE gibt es zwar (noch) kein offiziell unterstütztes KI-Plug-in von Tabnine. Der Hersteller hat aber mitgeteilt, dass er überlegt, ein solches zu entwickeln. Derzeit gibt es via GitHub nur ein inoffizielles Tabnine-Plug-in (siehe ix.de/zmtr).

Wer diese inoffizielle Version nicht einsetzen möchte, kann stattdessen auf das Plug-in von Codota zurückgreifen. Es lässt sich über den Eclipse-Marktplatz in der Version 1.0.14 (November 2020) herunterladen (siehe ix.de/zmtr). Hat man das Plug-in installiert, erhält man durch die Codevervollständigung von Eclipse weitere Vorschläge, die von Codota stammen und mit einem grünen Logo versehen sind. Die ersten Tests mit diesem Plug-in waren sehr vielversprechend.

IntelliJ IDEA mit Tabnine: Das KI-Tool macht hier einen syntaktisch falschen Vorschlag (blaues Icon); die Treffergenauigkeit hängt allerdings von der Datenbasis ab (Abb. 3).

Der Vorschlag von Tabnine, `erzeugen` zu verwenden, ist falsch. IntelliJ IDEA liefert hingegen an der zweiten Position den richtigen Methodenvorschlag (Abb. 4).

Geeignete Datenbasis ist Voraussetzung

Abbildung 3 zeigt ein Beispiel: Die Anweisung auf der rechten Seite der Abbildung soll erneut die Einträge eines Verzeichnisses im Dateisystem ermitteln und einem Array aus Elementen des Typs `String` zuweisen. Der erste Eintrag der Programmierhilfe von IntelliJ (rosa Icon) schlägt vor, die Methode `list()` zu verwenden. Dieser Vorschlag entspricht genau der Implementierung der Vorlage und ist syntaktisch korrekt. Der vierte Vorschlag von Tabnine (blaues Icon), stattdessen die Methode `listFiles()` zu verwenden, ist hingegen syntaktisch falsch, weil `listFiles()` ein Array mit Elementen des Typs `File` zurückliefert. Man kann Tabnine über die Eingabe von `Tabnine::sem` so konfigurieren, dass solche Fehler ausgeschlossen sind

Im zweiten Anlauf hat die KI-Erweiterung Tabnine gelernt und bringt ein besseres Ergebnis als die IntelliJ-interne Programmierhilfe (Abb. 5).

(siehe ix.de/zmtr). Trotzdem kam es in den Tests immer wieder zu syntaktisch falschen Vorschlägen.

Wie Tabnine mit fachlichen Klassen zurechtkommt, auf die es nicht angelernt werden konnte, sollte der abschließende Testlauf mit dem Java-Programm zur Bestellaufnahme untersuchen. Bei den fachlichen Klassen dieses Programms konnte Tabnine wie zu erwarten anfangs keine sinnvollen deutschen Namensvorschläge und Komplettierungen anbieten, da es dem Tabnine-Modell an einer geeigneten Datenbasis fehlt. Stattdessen ergänzte Tabnine die eingegebenen Anfänge der Bezeichner zu englischen Namen. Implementiert man die Klasse mehrmals erneut und versucht danach nochmals, die gleichen Attribute einzufügen, bemerkt man den Lerneffekt: Die Vorschläge entsprechen nun oftmals exakt der Implementierungsvorlage. Die Treffergenauigkeit der Vorschläge hängt also nur von den Modellen ab, mit denen Tabnine arbeitet.

Der weitere Verlauf der Implementierung erweist sich als durchwachsen. Die ersten Vorschläge für das Hauptprogramm mit den komplettiert implementierten fachlichen Klassen bringen häufig fehlerhafte Vorschläge. Hier wirkt sich negativ aus, dass Tabnine den Quellcode nicht analysiert, sondern „nur“ versucht, plausible Vorschläge auf Basis seiner Modelle anzuzeigen.

Ein Beispiel: Nimmt man den Tabnine-Vorschlag an, bei einem Objekt der Klasse `Bestellung` die Methode `erzeugen()` aufzurufen, führt das zu einem Fehler, weil diese Methode überhaupt nicht existiert. Der Vorschlag von IntelliJ IDEA, die Methode `erzeugeBestellposition(…)` aufzurufen, ist hingegen passgenau (Abbildung 4).

Es stellt sich die Frage, warum die unsichere Tabnine-Prognose an erster Stelle der Vorschläge steht und nicht der Vorschlag der Entwicklungsumgebung. Weiterhin ist unklar, warum an der Stelle des Rückgabewerts „tabnine“ steht. Das Tabnine-Icon zeigt ja bereits an, dass der Vorschlag von Tabnine stammt. Weil der Rückgabewert der Methode durch den Toolnamen verdeckt ist, fehlt eine wichtige Zusatzinformation. In vielen Fällen lässt sich nicht sofort erkennen, ob der Vorschlag syntaktisch richtig ist. Die große Lernfähigkeit der KI-Erweiterung zeigt sich aber, wenn man die Klasse korrekt fertigstellt und dann versucht, sie erneut zu schreiben. Tabnine schlägt eine Codevervollständigung vor, die deutlich besser ist als die der internen Programmierhilfe von IntelliJ (Abbildung 5).

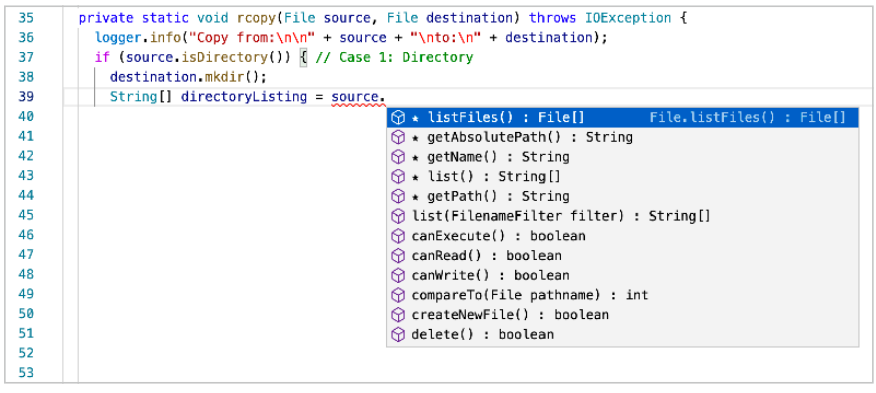
Ein Urteil über das KI-Tool Tabnine zu fällen, ist schwer. Es bringt ohne ausreichend angelerntes lokales Modell besonders bei technisch ausgelegten Programmen sofort einen Mehrwert, da die Trefferquote der Vorschläge dort relativ hoch ist. Bei fachlich orientierten Programmen muss man hingegen eine Lernphase in Kauf nehmen. Danach ergeben sich zum Teil erstaunlich gute Vorschläge. Man sollte allerdings keine Wunder erwarten. Unsichere Programmierer errettet das KI-Tool nicht vor den Untiefen der Syntax einer Programmiersprache, da die Erweiterung bei fehlerhaften Vorschlägen verwirren kann. Trotz dieses Mankos birgt die Technologie ein großes Potenzial. Es wird sich zeigen, wenn die Codebasis der zugrunde liegenden Modelle besser und umfangreicher wird.

Visual Studio Code mit IntelliCode

Visual Studio Code ist verglichen mit integrierten Umgebungen wie Eclipse IDE oder IntelliJ IDEA eher ein Editor. Microsoft hat das Tool Ende 2015 als Open Source freigegeben. Visual Studio Code ist für Windows, macOS und Linux verfügbar und kostenfrei. Es implementiert Programmiersprachen wie JavaScript, C++, C#, PHP, Python und Java. Das Installieren ist in wenigen Sekunden beendet. Für bestimmte Sprachen wie Java und Kotlin sind Erweiterungen zu installieren. Im Fall von Java stellt Red Hat sie bereit. Sie basieren auf den Eclipse-Technologien JDT, M2-Eclipse und Buildship (siehe ix.de/zmtr).

Der Test setzt Visual Studio Code 1.57.0 mit der Microsoft-KI-Erweiterung IntelliCode ein. Beim ersten Start findet Visual Studio Code die JRE automatisch, sofern sie korrekt installiert wurde. Der Editor präsentiert sich mit einer minimalistischen Oberfläche. Visual Studio Code arbeitet nicht mit Projektdateien, sondern – ähnlich der Eclipse IDE – mit einem Workspace-Konzept. Als Build-Tool lässt sich Maven einsetzen. Wie bei IntelliJ IDEA müssen sich Entwicklerinnen und Entwickler beim Erzeugen einer Klasse nicht durch mehrseitige Assistenten kämpfen. Stattdessen generiert Visual Studio Code zunächst eine leere Klassenhülle, die sich mit der Programmierhilfe vervollständigen lässt.

Der erste Testdurchlauf, das technische Kopierprogramm, produzierte Konstruktoren, Getter- und Setter-Methoden sowie Konstrukte wie Schleifen weitgehend automatisch. JavaDoc ergänzt Visual Studio Code automatisch, und auch das Kopieren



Visual Studio Code zeigt wie Tabnine die Methode `listFiles()` an (Abb. 6).

einzelner Abschnitte ist einfach. Möchte man eine Annotation einfügen, genügt es, ein @-Zeichen zu tippen, wie es bei vielen Editoren Standard ist.

Will man eine Methode eines Objekts aufrufen, zeigt der Editor alle möglichen Optionen an. Hier kann es aber zu Fehlgriffen kommen (Abbildung 6). Wie Tabnine bietet Visual Studio Code an erster Stelle die Methode `listFiles()` an, die an dieser Stelle einen Fehler verursachen würde. Deshalb den Algorithmus zu ändern, ist nicht zielführend.

Bei Try-Catch-Blöcken erzeugt Visual Studio Code den Rumpf des Blocks, wenn man das Schlüsselwort `try` eingibt. Es lässt sich darüber streiten, ob das sinnvoll ist. Wir empfinden das Angebot der Codevervollständigung hier als kontraproduktiv, denn es führt dazu, dass die Programmierhilfe eine unspezifizierte Exception im Catch-Block einsetzt, die Entwickler danach wieder mit einer korrekten Exception ersetzen müssen. Die Programmierhilfe kann schließlich nicht vor dem Eingeben der Methode, die eine Exception auslöst, ahnen, welche Exception sinnvollerweise zu behandeln ist. Es empfiehlt sich, wie bei IntelliJ zuerst die Anweisung einzugeben, die eine Exception werfen kann, und danach zu entscheiden, ob man sie weiterreicht oder an Ort und Stelle behandelt.

Beim fachlichen Test mit dem fiktiven Einkaufsprogramm traten beim Implementieren der Basisklassen keine Probleme auf. Allerdings ergaben sich auch keine nennenswerten Produktivitätsgewinne. Da ein Training der entsprechenden Klassen fehlt, lieferte die Codevervollständigung das, was IntelliJ IDEA zuvor ebenfalls angezeigt hatte. Vorschläge für fachlich sinnvolle Namen sind bei Visual Studio Code auch bei wiederholter Implementierung Fehlanzeige. Wie im technischen Test ließen sich Attribute anlegen und sowohl Konstruktoren als auch Getter- und Setter-Methoden erzeugen. Ein Lerneffekt war aber bei Visual Studio Code nicht erkennbar.

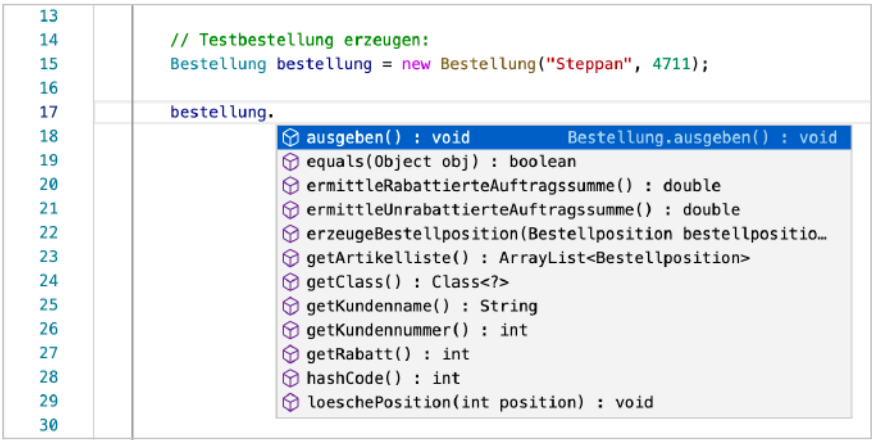
Wie schwer es für die KI-Tools ist, bei unbekannten Klassen sinnvolle Vorschläge anzuzeigen, zeigt Abbildung 7. Nach dem Implementieren der fachlichen Basisklassen `Artikel`, `Bestellung` und `Bestellposition` sollte eine Testklasse deren Funktion überprüfen. Ruft man die Methoden des Objekts `bestellung` auf, sieht es so aus, als ob Visual Studio Code empfehlen würde, die Bestellung auszugeben. Das erscheint unsinnig, da die Bestellung mit ihren Positionen noch nicht existiert. Schaut man sich die Vorschlagsliste genauer an, zeigt sich, dass die Vorschläge nicht nach Vorhersagegenauigkeit, sondern lediglich alphabetisch sortiert sind. Die KI-Erweiterung konnte noch keine Wissensbasis aufbauen, sinnvolle Vorschläge sind somit unmöglich. Alles in allem ist Microsofts Programmierhilfe IntelliCode im Vergleich zu Tabnine recht unauffällig. Die Vorschläge sind syntaktisch richtig, aber steigerten die Entwicklungsproduktivität bei unseren Tests nicht sonderlich.

Fazit

Welche der KI-Erweiterungen am besten geeignet ist, hängt von der Arbeitssituation ab. Besteht die Aufgabe darin, technische Klassen zu implementieren, ist Tabnine in Kombination mit IntelliJ IDEA oder VS Code eine lohnenswerte Ergänzung. VS Code konnte mit IntelliCode im Test im Vergleich mit Tabnine hingegen nicht vollständig überzeugen. Die Vorschläge, die IntelliCode angezeigt hatte, lagen zwar etwa auf dem Niveau von Tabnine. Es fehlte aber dessen Lernfähigkeit. Arbeitet man primär mit fachlichen Klassen, ist man mit der klassischen Codekomplettierung von IntelliJ IDEA ohne Erweiterungen oder VS Code ohne Erweiterungen schon sehr gut bedient.

Der Vergleich der drei Programmierhilfen zeigt, dass die Entwicklung gerade erst begonnen hat. Trotz des zwiespältigen Eindrucks, den Tabnine hinterlassen hat, birgt dieses Tool mit Sicherheit momentan das größte Potenzial. Es lässt sich mit vielen Entwicklungsumgebungen und Programmiersprachen einsetzen und lernt während des Programmierens sehr schnell.

(nb@ix.de)



IntelliCode zeigte bei fachlichen Klassen zwar syntaktisch richtige Vorschläge an, lieferte darüber hinaus aber keinen nennenswerten Produktivitätsgewinn (Abb. 7).

-Wertung

IntelliJ IDEA ohne Tabnine

- ⊕ sehr passende und syntaktisch korrekte Vorschläge
- ⊕ Community Edition kostenfrei
- ⊕ sehr gute Performance
- ⊖ keine weitreichende Codekomplettierung in Relation zu IntelliJ mit Tabnine oder VS Code mit IntelliCode
- ⊖ kein Lerneffekt im Testzeitraum feststellbar

IntelliJ IDEA mit Tabnine

- ⊕ weitreichende Codekomplettierung (verglichen mit IntelliJ IDEA ohne Tabnine)
- ⊕ Lerneffekt während des Testzeitraums feststellbar
- ⊕ Basisversion kostenfrei
- ⊕ sehr gute Performance
- ⊖ korrekte Vorschläge der klassischen Programmierhilfe werden manchmal durch unpassende Vorschläge von Tabnine verdrängt
- ⊖ teilweise syntaktisch falsche Vorschläge, darunter sogar Methoden, die in der entsprechenden Klasse nicht existieren

Visual Studio Code mit IntelliCode

- ⊕ weitreichende Codekomplettierung (verglichen mit IntelliJ IDEA ohne Tabnine)
- ⊕ kostenfrei
- ⊕ sehr gute Performance
- ⊖ kein Lerneffekt während des Testzeitraums feststellbar
- ⊖ korrekte Vorschläge der klassischen Programmierhilfe werden manchmal durch falsche Vorschläge von IntelliCode verdrängt
- ⊖ teilweise syntaktisch falsche Vorschläge

Quellen

KI-Funktionen, Plug-ins und weiterführende Informationen: ix.de/zmtr

Bernhard Steppan

arbeitet als Chefarchitekt bei DB Systel, dem Systemhaus der Deutschen Bahn.