

TDD und AI-Assistenten

Revolution oder Halluzination?
Java Forum Stuttgart 2025

andrena
OBJECTS

Gregor Wicklein
Gesine Kraft



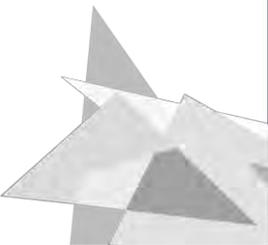
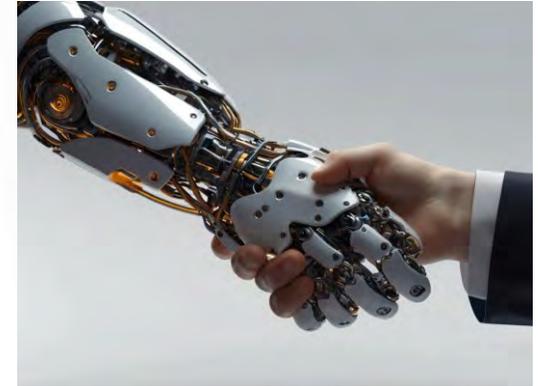
Klassische Methoden und AI-Assistenten

Bisher:

- Extreme-Programming
- Clean Code
- TDD
- Moderne IDEs
- Automatisierte Refactorings

Seit wenigen Jahren:

- LLMs
- AI-Codegenerierung
- Digitale Assistenten
- (Halb-)Autonome Agents



Was sind AI-Code-Assistenten?

- Smarte Code-Assistenten sind nicht neu
 - z.B. verwendet *Eclipse Code Recommenders* Machine Learning Modelle (Erstes Release 2012)
- Seit einiger Zeit werden jedoch verstärkt Generative-AI-Modelle genutzt
 - z.B. GitHub Copilot, JetBrains Junie Coding Assistant, Aider, Cody, Tabnine, codium, Windsurf, Cursor etc.
- Assistenten haben unterschiedliche Ansätze, z.B.
 - Code-Completion
 - Chat-Assistent
 - Inline-Assistent
 - Kombination unterschiedlicher Ansätze
 - AI-Extension für das IDE-Feature <fill-the-blank>

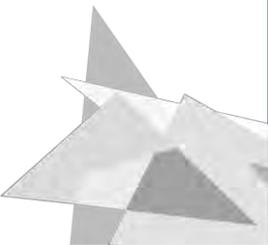


AI-Code-Completion

- User schreibt Code
- AI vervollständigt

```
public class Automobil {  
    private final String marke;  
    private final String modell;  
    private final int baujahr;  
    private final int kilometerstand;  
    private final int preis;  
  
    public Automobil(String marke, String modell, int baujahr, int kilometerstand, int preis) {  
        this.marke = marke;  
        this.modell = modell;  
        this.baujahr = baujahr;  
        this.kilometerstand = kilometerstand;  
        this.preis = preis;  
    }  
}
```

IntelliJ mit GitHub Copilot, Screenshot vom 23.05.2024



AI-Chat-Assistent

- User nutzt integriertes Chat-Fenster
- Assistent generiert Antwort im Chat

```
Write a Java class named "Automobil"
```

GitHub Copilot

> 3 steps completed successfully

Sure, here is a basic structure for a Java class named "Automobil". You can add more properties and methods as per your requirements.

```
public class Automobil {  
    // properties of the Automobil class  
    private String make;  
    private String model;  
    private int year;  
  
    // constructor  
    public Automobil(String make, String model, int year) {  
        this.make = make;  
        this.model = model;  
        this.year = year;  
    }  
}
```

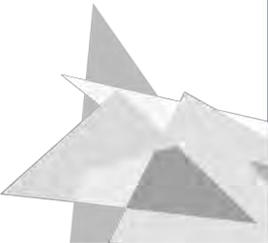
```
Schreibe eine Java-Klasse namens "Automobil"
```

GitHub Copilot

> 3 steps completed successfully

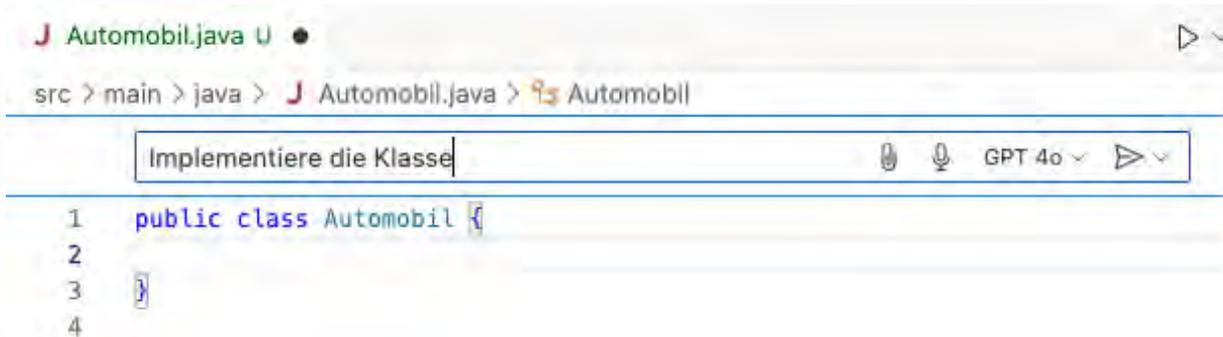
Sicher, hier ist eine grundlegende Struktur für eine Java-Klasse namens "Automobil". Sie können weitere Eigenschaften und Methoden hinzufügen, je nach Ihren Anforderungen.

```
public class Automobil {  
    // Eigenschaften der Klasse Automobil  
    private String hersteller;  
    private String modell;  
    private int baujahr;  
  
    // Konstruktor  
    public Automobil(String hersteller, String modell, int baujahr) {  
        this.hersteller = hersteller;  
        this.modell = modell;  
        this.baujahr = baujahr;  
    }  
}
```

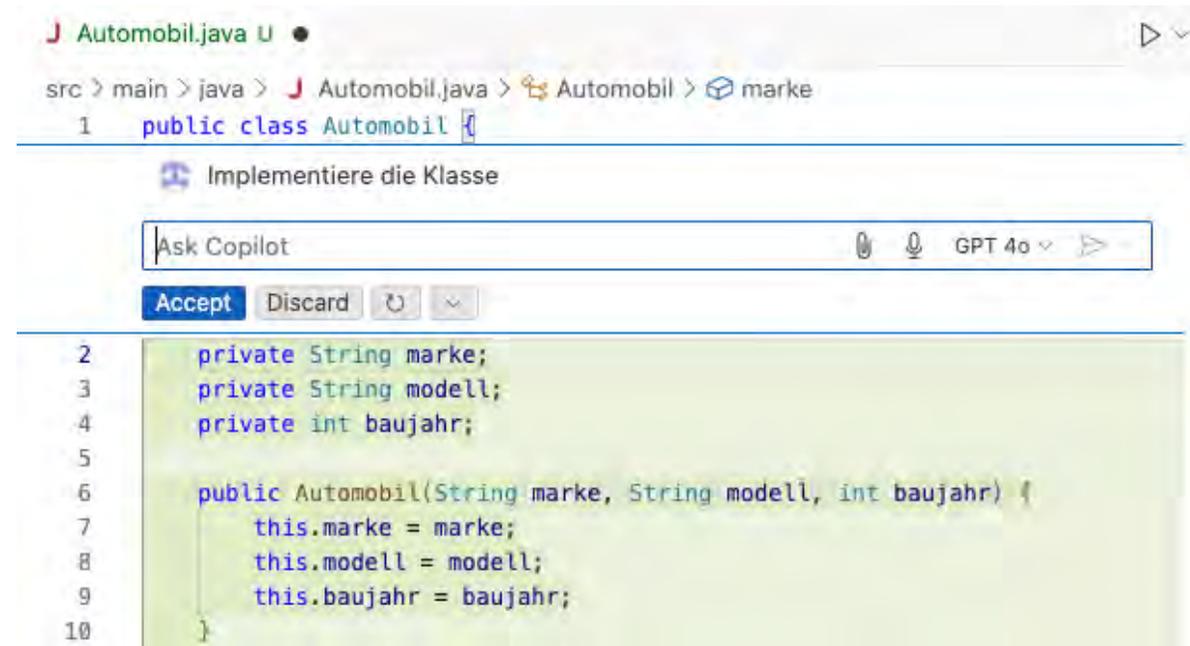


AI-Inline-Chat

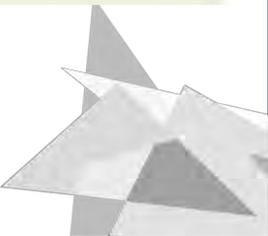
- User nutzt Chat im Code
- Assistent erstellt Vorschlag



The screenshot shows a VS Code editor window with the file 'Automobil.java' open. The breadcrumb navigation shows the path 'src > main > java > Automobil.java > Automobil'. A chat prompt 'Implementiere die Klasse' is entered in the inline chat interface. The code editor shows the start of a Java class definition: '1 public class Automobil {', '2', '3', and '4'.



The screenshot shows the same VS Code editor window, but now the inline chat interface has provided a suggestion. The prompt 'Implementiere die Klasse' is visible above the suggestion. The suggestion includes the text 'Ask Copilot' and buttons for 'Accept', 'Discard', and a refresh icon. The suggested code is highlighted in green and includes: '2 private String marke;', '3 private String modell;', '4 private int baujahr;', '5', '6 public Automobil(String marke, String modell, int baujahr) {', '7 this.marke = marke;', '8 this.modell = modell;', '9 this.baujahr = baujahr;', and '10 }'.

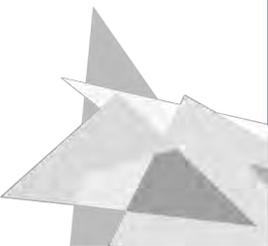


Kombination Chat-Assistent + Inline-Chat

- Anfrage in Chat-Assistent
- Generiert Code im Editor (ggf. in mehreren Dateien) (ggf. in mehreren Schritten)

The screenshot shows the Visual Studio Code editor with a Java file named 'Automobil.java'. The code defines a class with private attributes 'marke', 'modell', and 'baujahr', and public methods for getting and setting these attributes. On the right side, the GitHub Copilot chat interface is open, displaying a prompt in German: 'Erstelle eine Java-Klasse namens "Automobil" mit den Eigenschaften "marke", "modell" und "baujahr" sowie den entsprechenden Getter- und Setter-Methoden.' Below the prompt, it shows the generated code changes with a diff view (+30 -1). At the bottom of the chat, there are 'Accept' and 'Discard' buttons, and a 'Working Set (1 file)' section listing 'Automobil.java'.

```
1 public class Automobil {
2     private String marke;
3     private String modell;
4     private int baujahr;
5
6     // Getter und Setter für marke
7     public String getMarke() {
8         return marke;
9     }
10
11    public void setMarke(String marke) {
12        this.marke = marke;
13    }
14
15    // Getter und Setter für modell
16    public String getModell() {
17        return modell;
18    }
19
20    public void setModell(String modell) {
```



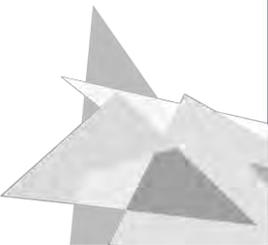
Halluzinationen

- LLMs erzeugen Texte indem sie jeweils das nächste Wort anhand der Wahrscheinlichkeit generieren.
- Allerdings nicht deterministisch (Stichwort: „Temperature“).
- Nur weil etwas wahrscheinlich ist, ist es nicht zwangsläufig korrekt.

```
public class Automobil {  
    // Eigenschaften der Klasse Automobil  
    private String hersteller;  
    private String modell;  
    private int baujahr;  
}
```

- Wer hat festgelegt, dass...
 - ...wir derartige Kommentare möchten?
 - ...Kommentare in Deutsch verfasst werden?
 - ...das Automobil diese Eigenschaften haben soll?
 - ...den Typ der Variablen?
 - etc.

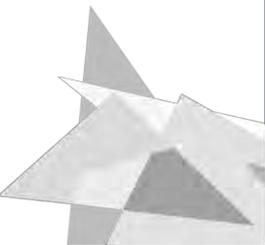
- Halluzinationen sind inkorrekte oder fiktive Informationen, die ein LLM generiert.



Code-Assistent – Kontext (1)

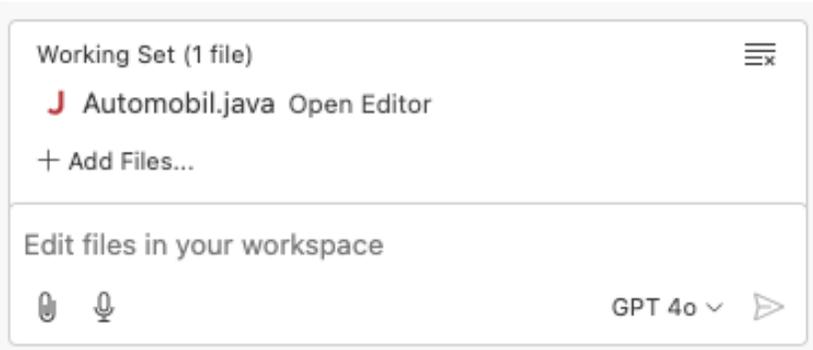
- Problem:
 - Large Language Model (LLM) ist fertig trainiert
 - Für kontext-sensitives Coding ist Kontext (Dateien, Anforderungen etc.) notwendig
 - Gesamter Workspace ist i.d.R. zu groß
 - (zu teuer)
 - zu unspezifisch, daher schlechtere Ergebnisse [1]
- Lösungsansätze
 - Ein Teil der Code-Basis wird als sogenannter Kontext zur Verfügung gestellt
 - Unterschiedliche Lösungsansätze
 - Heuristische Auswahl
 - Explizite Auswahl
 - Suche und Auswahl über indizierte Codebasis

[1] <https://arxiv.org/abs/2402.14848>



Code-Assistent – Kontext

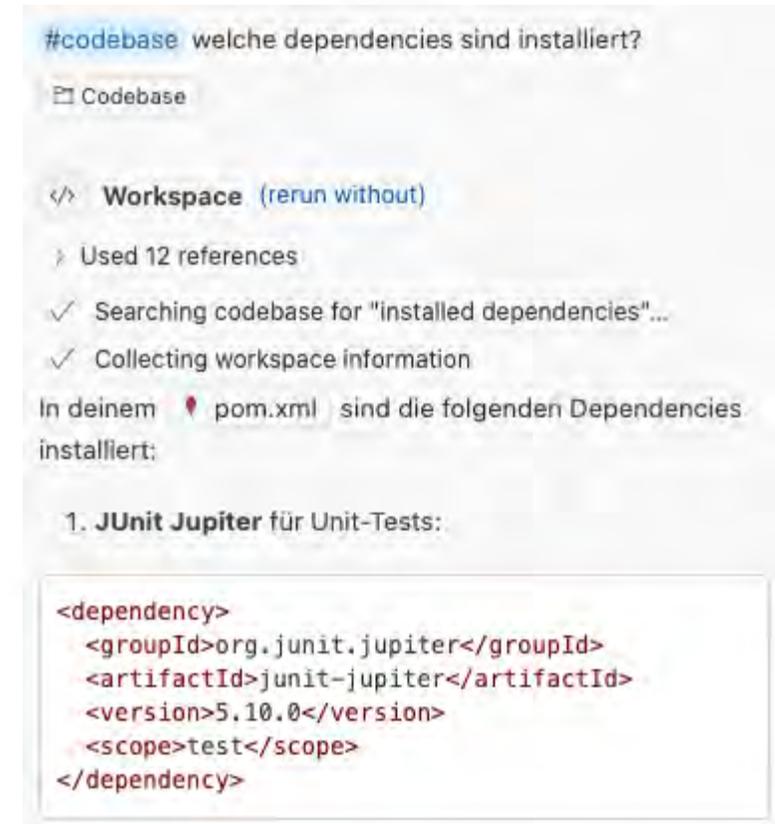
● Explizite Auswahl



● Heuristische Auswahl



● Suche und Auswahl über indizierte Codebasis



Motivation für AI-Assistent + TDD

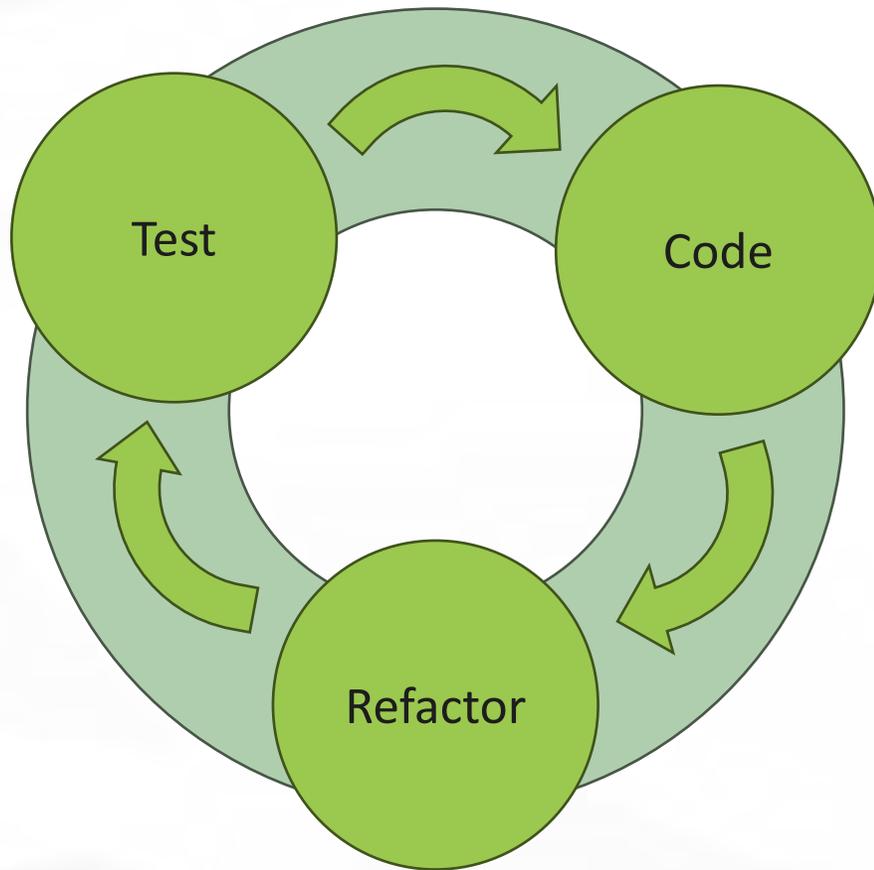
- Code-Assistenten halluzinieren zwangsläufig
 - Kontext unvollständig
 - Anforderungen nicht oder nicht vollständig bekannt
 - Code-Completion stellt keine Rückfragen

- Wie dokumentieren wir Anforderungen im Code am besten?
=> Durch ausführbaren, selbstvalidierenden Code, ergo automatisierte Tests

- Idee:
 - Wenn wir erst Tests schreiben, stehen die Anforderungen im Kontext
 - Erfüllung der Anforderungen lässt sich über Tests überprüfen
 - Vermeidung von Folgefehlern

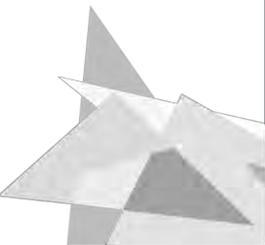


Test-Driven Development (TDD)

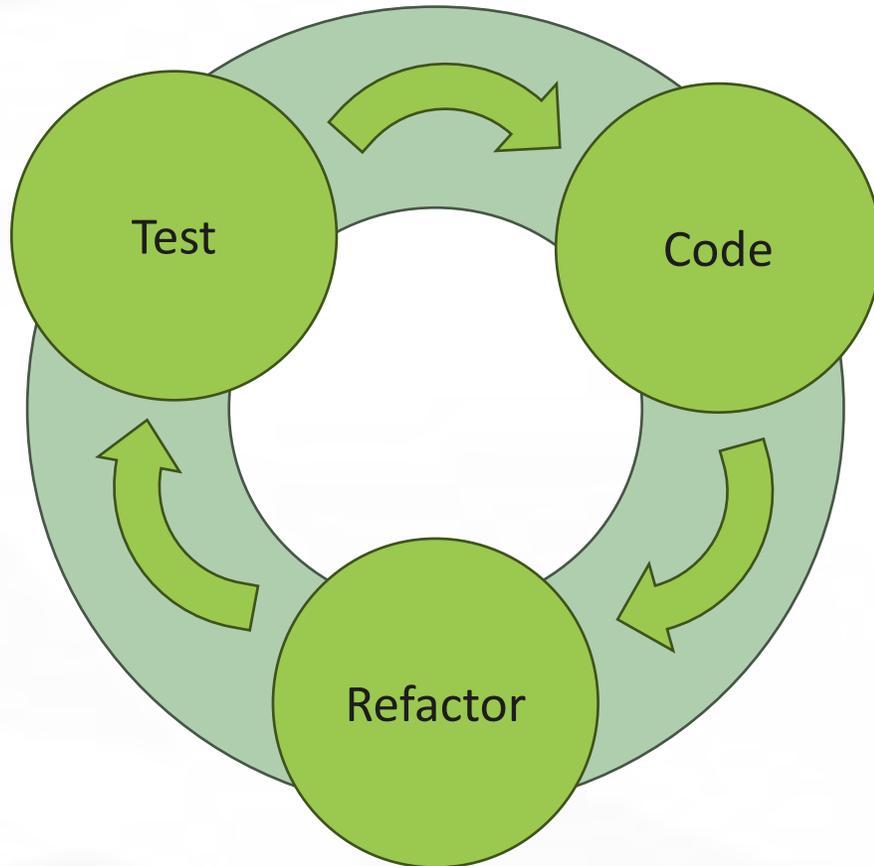


Regeln:

- Schreibe keinen produktiven Code bevor du nicht einen roten Test hast
- Schreibe nur so viel Test-Code, dass der Test fehlschlägt
- Schreibe nur minimal viel produktiven Code, um den Test grün zu machen.
- Wenn alle Tests grün sind, räume den Code auf (refactor)

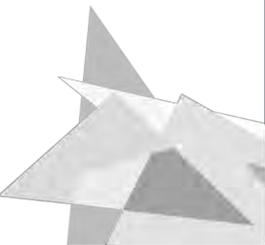


Test-Driven Development (TDD)



Vorteile:

- Hohe Testabdeckung
- Saubere und gut testbare Architektur
- Gute/Lesbare Tests = Dokumentation



Unser Setup heute: Cursor mit Claude Sonnet 3.5

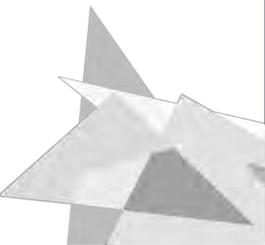
Intellij

- + GitHub Copilot mit Claude 3.7
- Erzielt gute Ergebnisse in aktuellen Coding Benchmarks

Altes Setup

- Cursor + GitHub Copilot
- IntelliJ + GitHub Copilot
- VS Code + GitHub Copilot

- Disclaimer: Rasante Entwicklung, unzählige Tools, wir müssen eine Auswahl treffen. Vieles ist auch auf andere IDEs und Assistenten übertragbar.

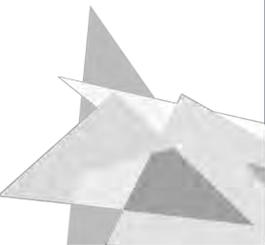




Live-Coding

AI-Unterstütztes TDD: Die Stärken

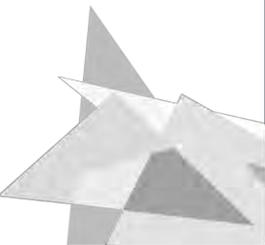
- Schrittweises Vorgehen hilft bei der Überprüfung des generierten Codes
- Schnelle Generierung von Tests und Boilerplate-Code
- Teilweise passende Vorschläge
- Sicherer Umgang mit Libraries und (Test)Frameworks
- Gute Erläuterungen und Kontextualisierung von bestehenden und generierten Code
- Weniger Kontext-Switch, weniger Wechsel aus der IDE (z.B. weniger Suchmaschine)
- Insgesamt mehr Flow möglich



AI-Unterstütztes TDD: Die Schwächen

- Auswahl von Testwerten durchwachsen, Schwierigkeiten mit Randfällen
- Fehleranfällig beim Rechnen
- AI-Refactoring ist mit Risiko verbunden, d.h. kein "sicheres refactoring"
 - Besser: IDE-Tools kennen und nutzen
- Keine/wenig Ausführung von IDE-Features (z.B. Refactoring oder Testausführung)

- Seit einiger Zeit besser geworden:
 - Auswahl von Testwerten durchwachsen
 - Probleme durch limitierten Kontext (Verbesserung z.B. durch bessere Auswahl)
 - Teils sinnlose Permutationen von Testfällen
 - Bearbeitung von mehreren Dateien

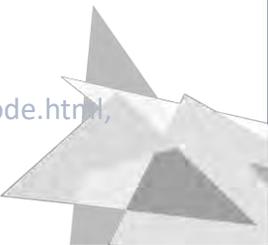


Wo stehen wir? Vergleich zum autonomen Fahren

- Um die Rolle des Fahrenen zu verdeutlichen, unterscheidet die Bundesanstalt für Straßen- und Verkehrswesen drei Modi

Assistierter Modus	Automatisierter Modus	Autonomer Modus
SAE Level 1 und 2	SAE Level 3	SAE Level 4 und 5
Mensch hat permanent die Verantwortung	<ul style="list-style-type: none"> System hat in bestimmten Situationen die Verantwortung; Mensch muss nach Aufforderung übernehmen 	System hat die gesamte Verantwortung (bei Level 4 auf gewissen Strecken)
ACC, Spurhalteassistent Level 2: Kombination	Staupilot (z.B. Autobahn, rechte Spur bis 95 km/h)	Autonomes Parkhaus Level 5: Autonome Busse

Darstellung in Anlehnung an Kraftfahrt-Bundesamt gemäß dreistufiger Unterteilung der Bundesanstalt für Straßen- und Verkehrswesen, https://www.kba.de/DE/Themen/Marktueberwachung/Produktpruefungen/AutomatisiertesAutonomesFahren/Automatisierungsstufen/Automatisierungsstufen_node.html, abgerufen am 04.02.25



Perspektive (Meinung)

- Ausführung von IDE-Features, z.B.
 - Tests ausführen und Ergebnis verarbeiten
 - Automatisches, sicheres Refactoring durchführen
 - Debugger nutzen
- Mehr customized Agents
 - Debugging Agent
 - Code Review Agent
 - Testing Agent
 - Code Migration Agent
 - Documentation Agent
 - <your-fancy-idea> Agent
 - z.B. bietet GitHub Copilot Extensions an



Copilot Extensions

Extend Copilot capabilities using third party tools, services, and data

[Preview](#) [Give feedback](#)

Filter: All | By: All creators | Sort: Popularity

 Docker for GitHub Copilot Copilot Learn about containerization, generate Docker assets and analyze project vulnerabilities in GitHub Copilot	 PerplexityAI Copilot Perplexity answers questions as you code by searching the web
 Stack Overflow Copilot Get answers to your most complex coding questions right where you're already working	 Models (GitHub) Copilot Copilot Extension to connect and chat with GitHub Models
 Mermaid Chart Copilot Provides advanced and powerful diagramming and visualization to GitHub Copilot Chat	 ReadMe API Copilot Ask questions about the ReadMe API and get help with code, directly in VS code

Symbolbild, generiert mit DALL-E 3

GitHub Copilot Marketplace, Screenshot vom 3.2.25

Pair Programming vs. AI-Code-Assistent

- Beim klassischen Pair Programming entwickeln zwei Personen gemeinsam an einem Rechner
- Welche Aspekte könnte ein AI-Code-Assistent theoretisch erfüllen?
 - Fehlererkennung
 - Höhere Codequalität
 - Schnellere Problembeseitigung
 - Schnelleres Feedback
 - Lerneffekt (beschränkt auf eine Person)
- Welche Aspekte kann ein AI-Code-Assistent nicht erfüllen
 - Kollaboration / Förderung von Teamarbeit
 - Isolation entgegenwirken
 - Wissenstransfer zwischen Teammitgliedern
 - Risikoreduzierung durch verteiltes Wissen
- Welche Vorteile könnte ein AI-Code-Assistent bieten
 - Geringere Kosten, höhere Geschwindigkeit, Skalierbarkeit, Verfügbarkeit



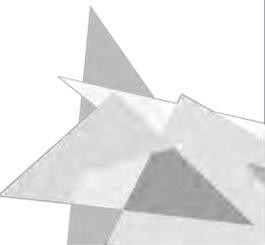
Symbolbild, generiert mit DALL-E 3

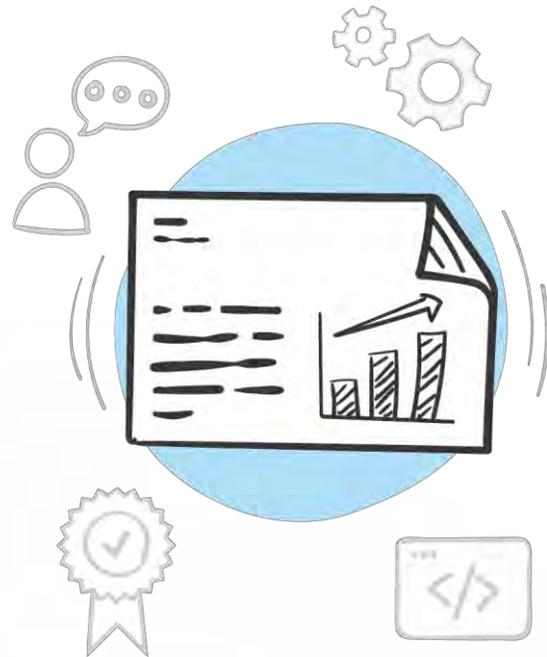
Takeaways

- The assistant is your copilot → You are the pilot
- доверяй, но проверяй (Trust but verify)
- Choose the right tool for the right job



Stay tuned for more...





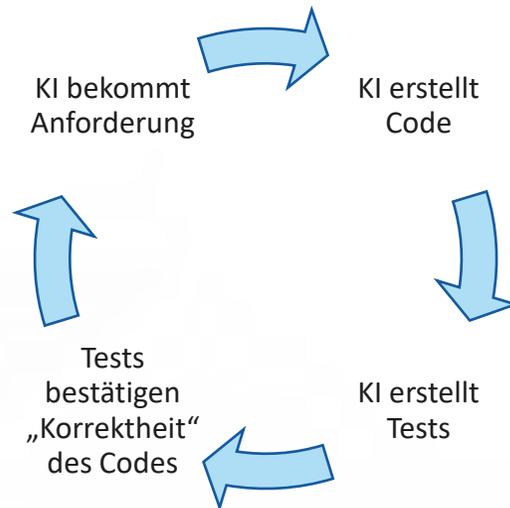
Backup-Folien

TDD vs. Test last – im Kontext von AI-Assistenten



Junior Developer

„Cool, ich kann mir alle Tests generieren lassen!“



- Angenommen die KI halluziniert...

- Erstellter Code enthält Fehler
- Tests bestätigen Fehler
- Fehler bleibt unentdeckt
- Zukunfts-Dev:
„Das scheint eine Anforderung zu sein“

=> Tests spiegeln Anforderungen nicht wider

