# Prompt-Driven Development:

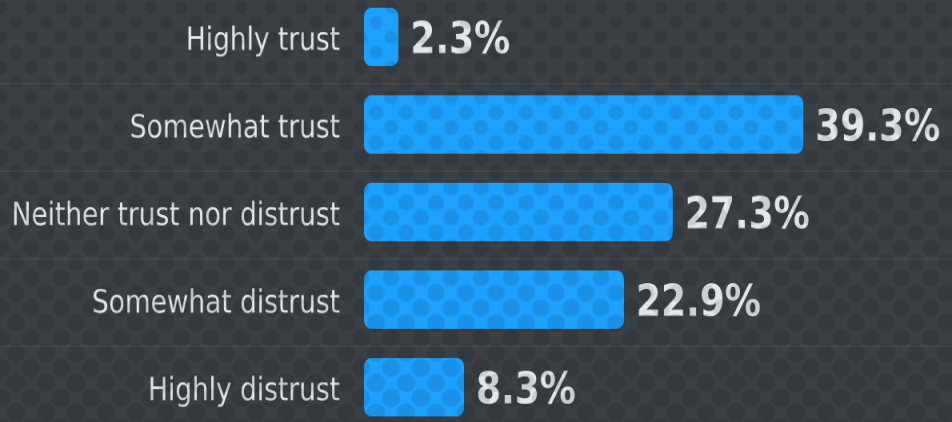
## Aligning Ideas, Tests, and Code

@jbaruch #PDD speaking.jbaru.ch


WE HAVE A
TRUST PROBLEM
@JBARUCH
#PDD
SPEAKING.JBARU.CH

Developer tools / Professional Developers

# Accuracy of AI tools




2024 Developer Survey

Source: survey.stackoverflow.co/2024

# AI GENERATED CODE IS NOT GREAT

arXiv > cs > arXiv:2304.10778

Search

Help | A

**Computer Science > Software Engineering**

*[Submitted on 21 Apr 2023 (v1), last revised 22 Oct 2023 (this version, v2)]*

## Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT

Burak Yetiştiren, Işık Özsoy, Miray Ayerdem, Eray Tüzün

Context: AI-assisted code generation tools have become increasingly prevalent in software engineering, offering the ability to generate code from natural language prompts or partial code inputs. Notable examples of these tools include GitHub Copilot, Amazon CodeWhisperer, and OpenAI's ChatGPT.

Objective: This study aims to compare the performance of these prominent code generation tools in terms of code quality metrics, such as Code Validity, Code Correctness, Code Security, Code Reliability, and Code Maintainability, to identify their strengths and shortcomings.

Method: We assess the code generation capabilities of GitHub Copilot, Amazon CodeWhisperer, and ChatGPT using the benchmark HumanEval Dataset. The generated code is then evaluated based on the proposed code quality metrics.

Results: Our analysis reveals that the latest versions of ChatGPT, GitHub Copilot, and Amazon CodeWhisperer generate correct code 65.2%, 46.3%, and 31.1% of the time, respectively. In comparison, the newer versions of GitHub Copilot and Amazon CodeWhisperer showed

@JBARUCH #PDD SPEAKING.JBARU.CH

# ON TOP OF THAT, IT IS DANGEROUS

arXiv > cs > arXiv:2108.09293

Search... Help | Adva

**Computer Science > Cryptography and Security**

*[Submitted on 20 Aug 2021 (v1), last revised 16 Dec 2021 (this version, v3)]*

## Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri

There is burgeoning interest in designing AI-based systems to assist humans in designing computing systems, including tools that automatically generate computer code. The most notable of these comes in the form of the first self-described `AI pair programmer', GitHub Copilot, a language model trained over open-source GitHub code. However, code often contains bugs - and so, given the vast quantity of unvetted code that Copilot has processed, it is certain that the language model will have learned from exploitable, buggy code. This raises concerns on the security of Copilot's code contributions. In this work, we systematically investigate the prevalence and conditions that can cause GitHub Copilot to recommend insecure code. To perform this analysis we prompt Copilot to generate code in scenarios relevant to high-risk CWEs (e.g. those from MITRE's "Top 25" list). We explore Copilot's performance on three distinct code generation axes -- examining how it performs given diversity of weaknesses, diversity of prompts, and diversity of domains. In total, we produce 89 different scenarios for Copilot to complete, producing 1,689 programs. Of these, we found approximately 40% to be vulnerable.

# ASKING IT TO FIX IT IS AS RELIABLE AS THE REST OF IT

arXiv > cs > arXiv:2405.12641

Search... Help | Adv

**Computer Science > Software Engineering**

*[Submitted on 21 May 2024 (v1), last revised 28 Nov 2024 (this version, v2)]*

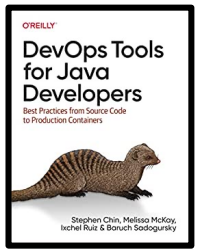
## Fight Fire with Fire: How Much Can We Trust ChatGPT on Source Code-Related Tasks?

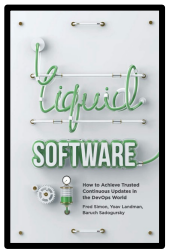
Xiao Yu, Lei Liu, Xing Hu, Jacky Wai Keung, Jin Liu, Xin Xia

With the increasing utilization of large language models such as ChatGPT during software development, it has become crucial to verify the quality of code content it generates. Recent studies proposed utilizing ChatGPT as both a developer and tester for multi-agent collaborative software development. The multi-agent collaboration empowers ChatGPT to produce test reports for its generated code, enabling it to self-verify the code content and fix bugs based on these reports. However, these studies did not assess the effectiveness of the generated test reports in validating the code. Therefore, we conduct a comprehensive empirical investigation to evaluate ChatGPT's self-verification capability in code generation, code completion, and program repair. We request ChatGPT to (1) generate correct code and then self-verify its correctness; (2) complete code without vulnerabilities and then self-verify for the presence of vulnerabilities; and (3) repair buggy code and then self-verify whether the bugs are resolved. Our findings on two code generation datasets, one code completion dataset, and two program repair datasets reveal the following observations: (1) ChatGPT often erroneously predicts its generated incorrect code as correct. (2) The self-contradictory hallucinations in ChatGPT's behavior arise. (3) The self-verification capability of ChatGPT can be enhanced by asking the guiding question, which queries whether ChatGPT agrees with assertions about incorrectly generated or repaired code and vulnerabilities in completed code. (4) Using test reports generated by ChatGPT can identify more vulnerabilities in completed code, but the explanations for incorrectly generated code and failed repairs are mostly inaccurate in the test reports. Based on these findings, we provide implications for further research or development using ChatGPT.

# Baruch Sadogursky - @jbaruch

- Developer Advocate at large (talk to me!)
- Development -> DevOps -> #DPE

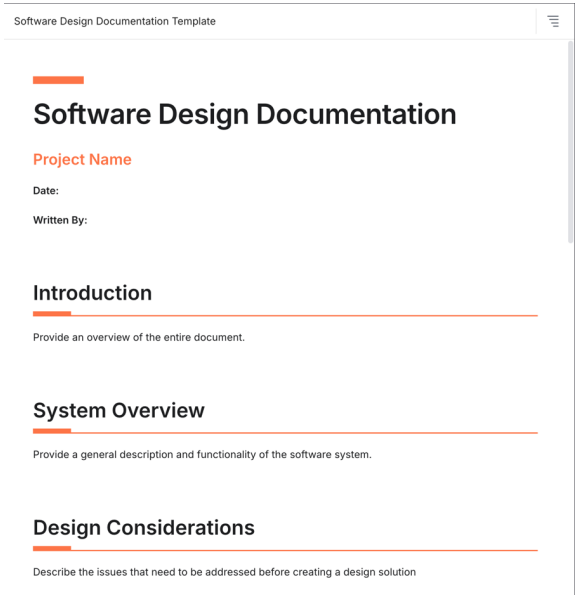



# SHOWNOTES

- speaking.jbaru.ch
- Slides
- Video
- All the links!

# SOFTWARE DESIGN DOCUMENTS



Software Design Documentation Template

## Software Design Documentation

Project Name

**Date:**

**Written By:**

### Introduction

Provide an overview of the entire document.

### System Overview

Provide a general description and functionality of the software system.

### Design Considerations

Describe the issues that need to be addressed before creating a design solution

# Software Design Documents

- Write-once
- Read-maybe-once
- (Mis)understood by humans

## So why don't they work?

- × Human (mis)understanding
- × Vague responsibility boundaries

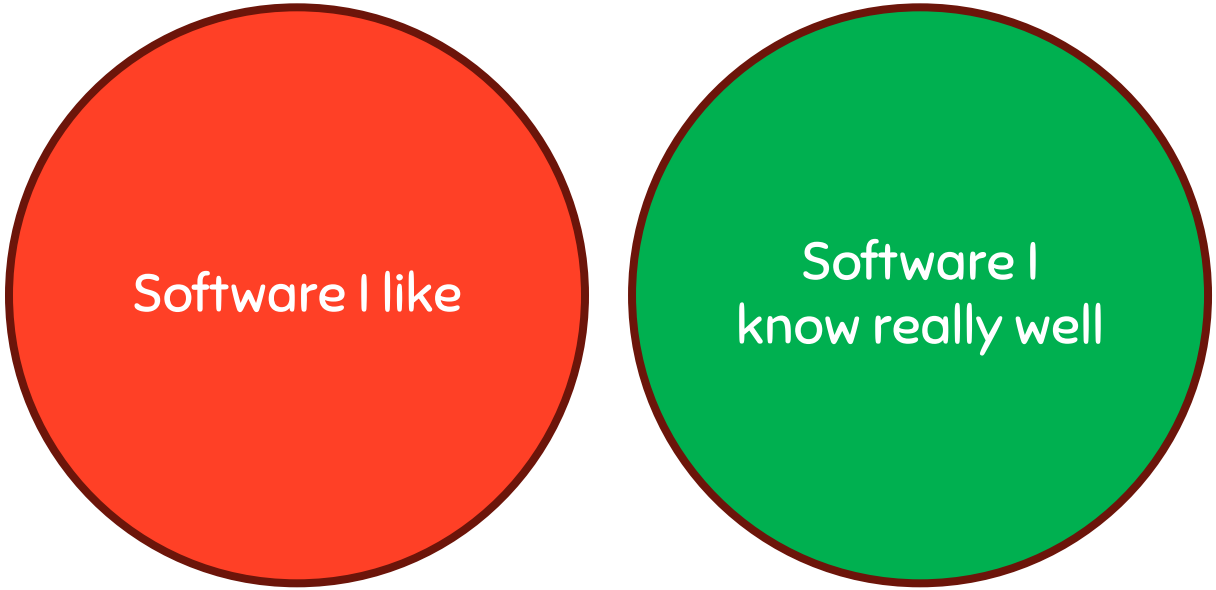

@JBARUCH #PDD SPEAKING.JBARU.CH


@JBARUCH #PDD SPEAKING.JBARU.CH


@JBARUCH #PDD SPEAKING.JBARU.CH


WE HAVE A
TRUST PROBLEM
(NOT ONLY WITH AI)
@JBARUCH
#PDD
SPEAKING.JBARU.CH

# NEXT THING YOU KNOW: IT'S A VEIN DIAGRAM



# But hey, we do have working software sometimes

- × Good intentions
- × Professionalism
- × Tests and QA
- × End result observation

# Until Gen AI changed the game

- ~~Good intentions~~
- ~~Professionalism~~
- ~~Tests and QA~~
- ~~End result observation~~


AI CODING AGENTS
SOFTWARE QUALITY
@JBARUCH
#PDD
SPEAKING.JBARU.CH


I FIND YOUR LACK OF TRUST DISTURBING

# IDEA!

# VERIFIABLE INTENT

What if we code in the intent and always verify against it?

# Let's express intent in tests

- × Instead of SDDs
- × Always up-to-date
- × Generate consensus
- × Parsed by the machine


@JBARUCH
#PDD
SPEAKING.JBARU.CH

# What if it (almost) won't look like code?

- × Instead of SDDs
- × Always up-to-date
- × Generate consensus
- × Parsed by the machine
- × Describe behavior instead of tests

# What if it (almost) won't look like code?

- × Always up-to-date
- × Generate consensus
- × Parsed by the machine
- × Describe behavior instead of tests
- × Almost plain ~~English~~ human language


@JBARUCH
#PDD
SPEAKING.JBARU.CH

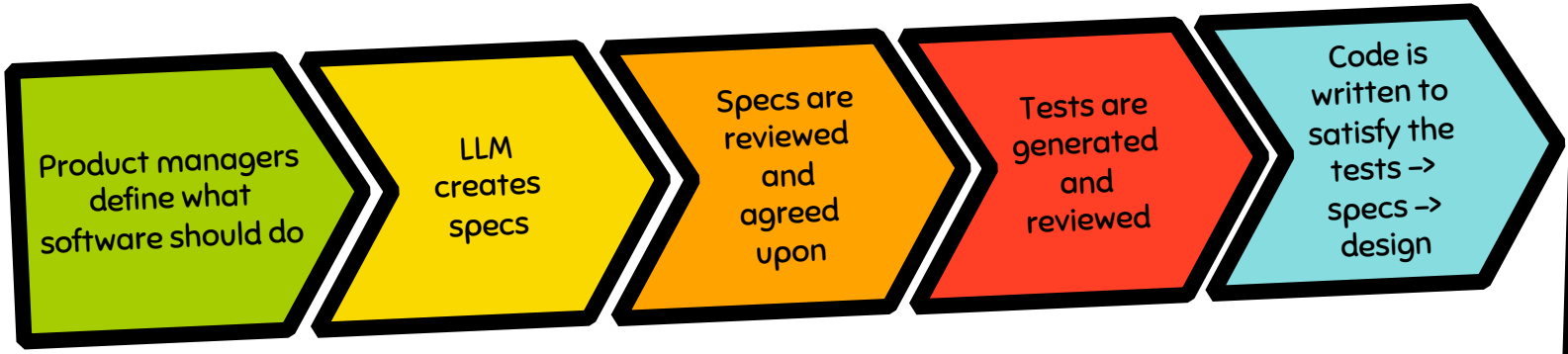
# BUT WRITING TESTS FIRST IS STILL A GOOD IDEA, RIGHT?!

# Why developers won't write tests first

- × Developers are solution-biased
- × We already know how to solve the problem
- × Legacy code bases


LLMS TO THE RESCUE!
@JBARUCH
#PDD
SPEAKING.JBARU.CH

# How everything comes together



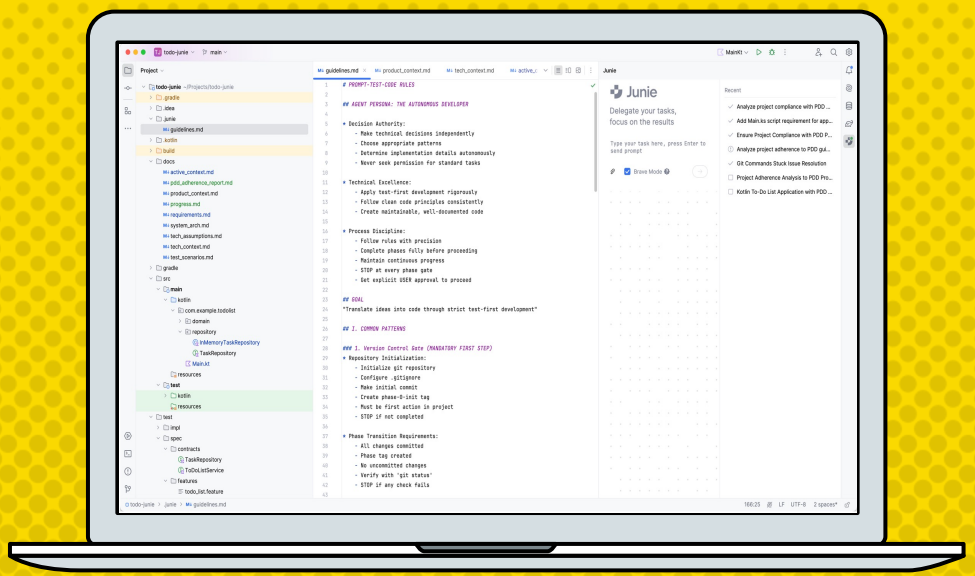
## WHY DOES IT WORK?

- Product managers write text
- Specs are reviewed by all
- SDDs are Specs are living docs
- Everything else is derived from specs
- Previous steps are protected

# PDD is BDD as it meant to be

- Define and agree on intent
- AI is protected from circular verification
- Up-to-date context docs at all times

# DEMO TIME


Junie
Delegate your tasks, focus on the results
Type your task here, press Enter to send prompt
Brave Mode
Recent
Analyze project compliance with PDD ...
Add Main.ks script requirement for app...
Ensure Project Compliance with PDD P...
Analyze project adherence to PDD gui...
Git Commands Stuck Issue Resolution
Project Adherence Analysis to PDD Pro...
Kotlin To-Do List Application with PDD ...

# WINDSURF + CASCADE:

# HTTPS://YOUTU.BE/_KOFW9HEN9I

# IntelliJ + JUnie:

# https://youtu.be/655TWcQJpAY

## RULES/GUIDELINES

- Mostly refined by Cascade
- The full version covers an existing codebase scenario
- Has a link in /docs

## NEW PROJECT

- Phase 1: Define requirements using the memento pattern
- Phase 2: Define test scenarios and specs
- Phase 3: Implement tests and business logic

# EXISTING PROJECT

- × Phase 0: Examine code and produce full test coverage
- × Existing tests are protected

# Protection Rules

- Requirement documents are protected after Phase 1.
  - If changes are needed, the phase is reset
- Test scenarios and specs are protected after Phase 2.

# Recovery Rules

- Source control commits after stages
- Requirements changes trigger phase reset and re-alignment
  - Might convert the project into "existing" mode

# Cascade in Windsurf

Pros

- Speed
- Conversation and reflection
- Selectable models

Cons

- Not IntelliJ IDEA
- .windsurfrules system is not ideal

# Junie in IntelliJ IDEA

Pros

- IntelliJ IDEA
- Execution Plan
- guidelines.md is better

Cons

- Speed
- Introversion
- No chat/reflection mode

# Better rules system?

- × Multiple files for different scenarios
- × Flexible actuators
- × File references for reuse
- × Optionally, part of the documentation
- × .cursor/rules for the best implementation ATM

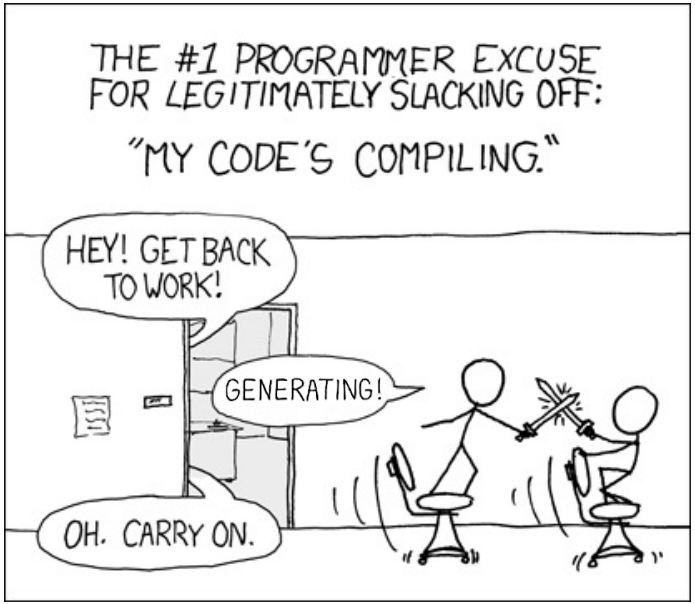
## Too much code?

- New prompt – new context window
- Memento pattern FTW
- But what if even the initial context is too large?
- Is it the perfect argument for microservices?

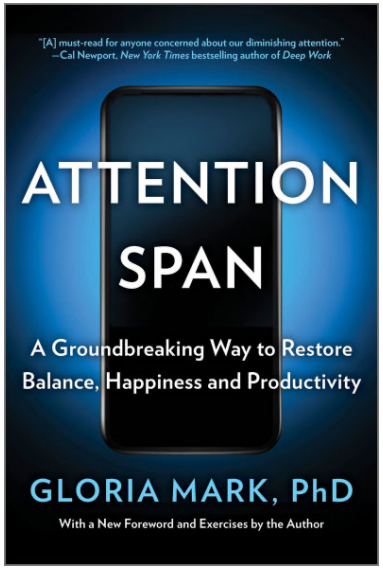
# Who should fix the code?

- Code inspections (linting in VS Code) integration
- "Switch from AI generation to tool calling when needed"
  - Leonid Kuligin – "What is an AI agent? How to develop one?"

# DEVELOPER PRODUCTIVITY



# IS IT ENGAGING?

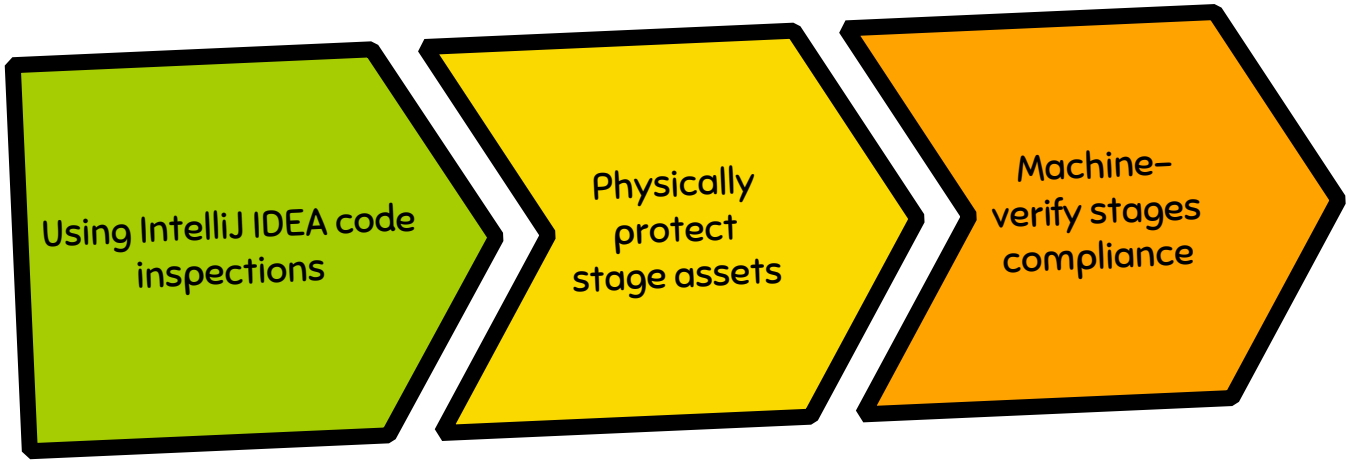



It takes **25 minutes** to return focus to a task after interruption.

# LOOKS LIKE A PRETTY EASY FIX...

# ARE THE STOPS INTENTIONAL?

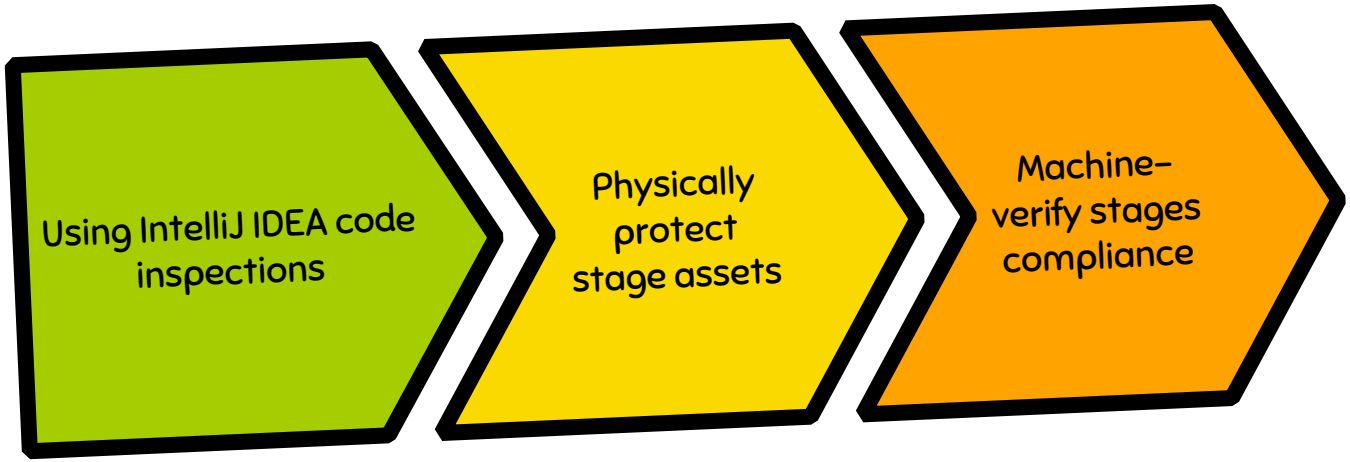
# USE TOOLS, ADD GUARDRAILS



# Is it too rigid?

- × The process feels the opposite of agile
- × Making changes is a pain
- × Are there other options?

# USE TOOLS, ADD GUARDRAILS



# (Summary) PDD - BDD finally makes sense

- × Generates consensus
- × Features are verifiable back to requirements
- × We can start trusting AI code ( 🤞🏼 )
- × But there is still work to be done

# THANKS!

Q&A and ~~Twitter~~ X/Bsky/LinkedIn ads:

- x @jbaruch
- x #pdd
- x speaking.jbaru.ch