

An Test-driven Development gescheitert? Hier erfährst du, warum.

Fallen und Missverständnisse vermeiden



Warum bis **du** hier ?

Was Ihr mitbringen solltet

- Verständnis der Grundbegriffe von TDD
- mindestens mal ein Tutorial mit einem praktische Beispiel gemacht
- Durchhaltevermögen auch bei schwierigen Situationen
- keine Pläne noch jahrelang in einem Legacy - Projekt ohne Testabdeckung zu arbeiten

Was euch erwartet

- hauptsächlich theoretischer Vortrag
- nur ein Codebeispiel
- kleiner Einblick in KI-gestütztes TDD -Entwicklung

Wie viel TDD - Praxis hast du ?

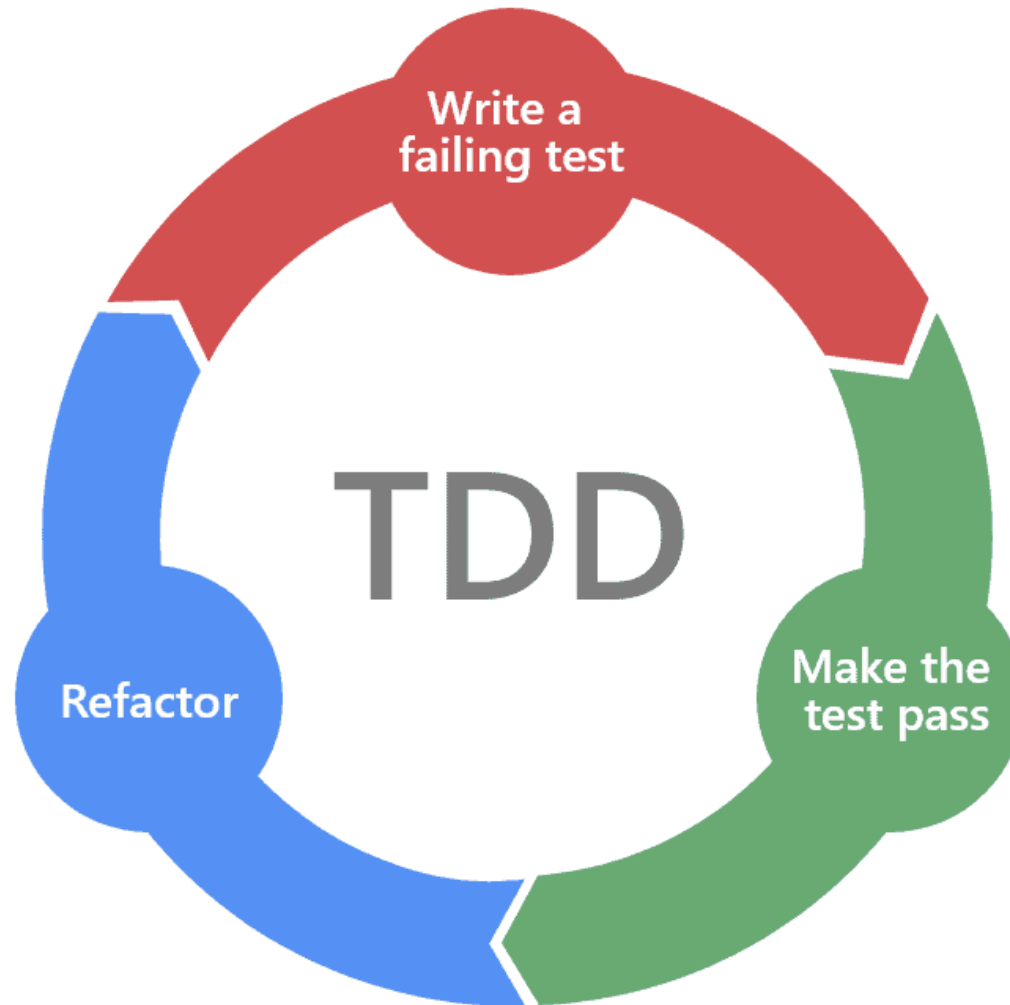
Mentimeter



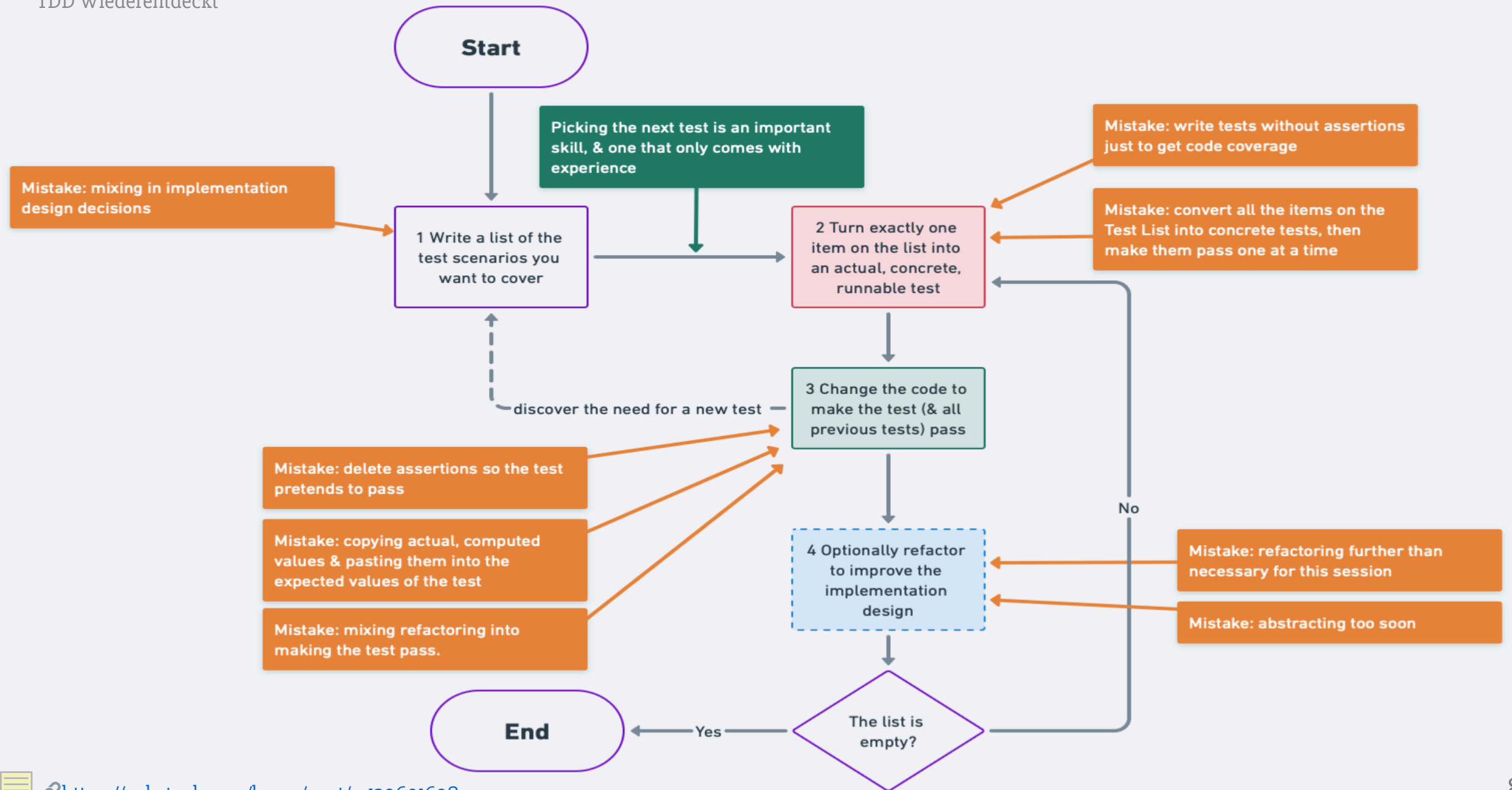
Wer ich bin:



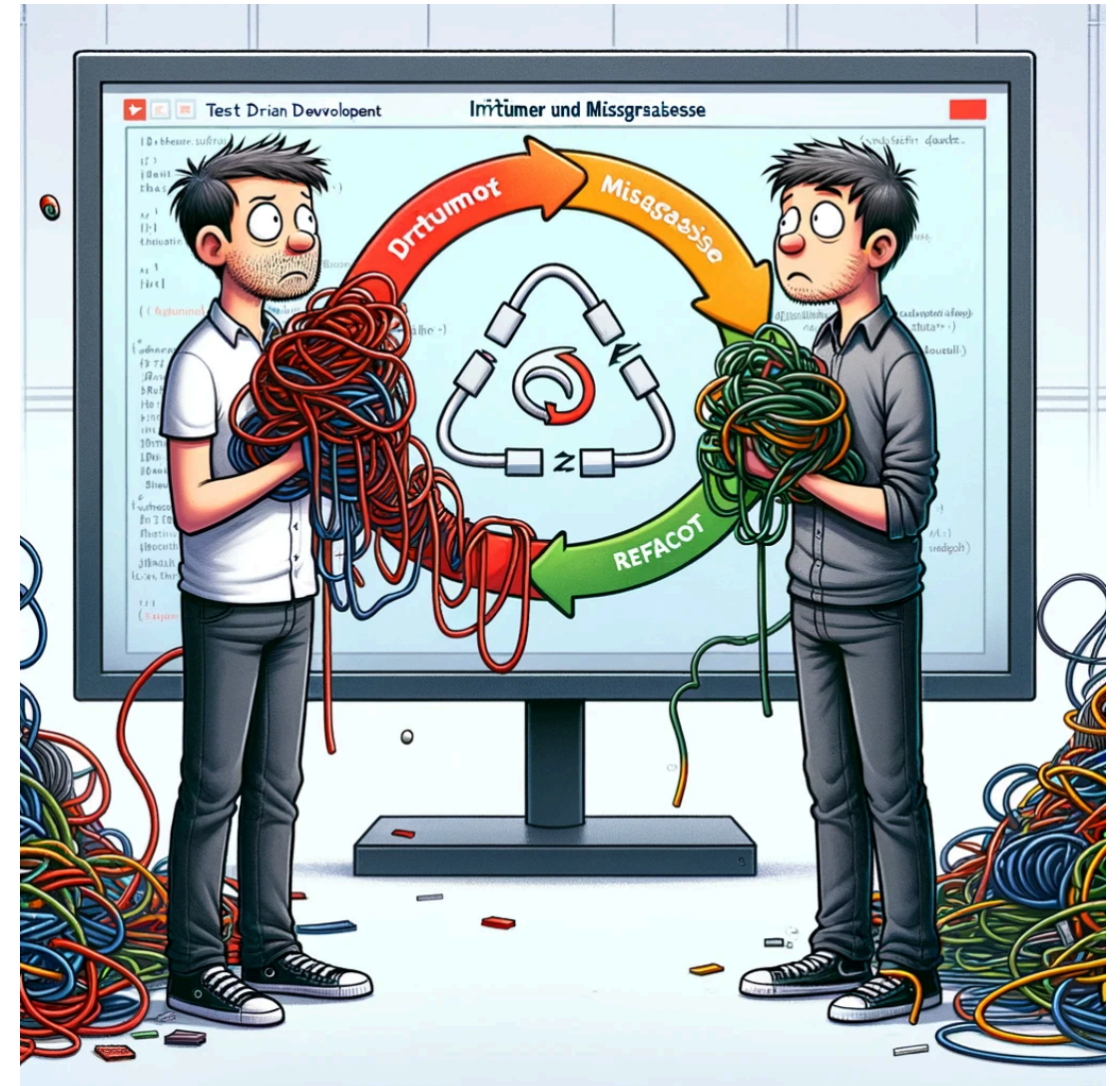
- Bodo Teichmann
- Softwareentwickler seit mehr als 30 Jahren
- "early adopter" von TDD
- 6 Jahre `C` Erfahrung, 10 Jahre `Java`
- CI-Automatisierung, ein wenig `Angular`, diverse Script Sprachen,
- zuletzt 3 Jahre `Spring Boot`



**TDD, ganz
einfach ?**



7 Irrtümer von TDD



.. oder "Über 7 Brücken
musst du gehen, 7 dunkle
Jahre überstehen"

▶ 0:00 / 0:28 — 🔊 ⋮



1. „Unit“ Testing

3. TDD ist unflexibel

5. TDD ist nur für erfahrene Entwickler

7. BDD / der PO schreibt jetzt die Tests

2. Verwende viele Mocks

4. TDD kostet Zeit

6. Namen von Testmethoden sind egal

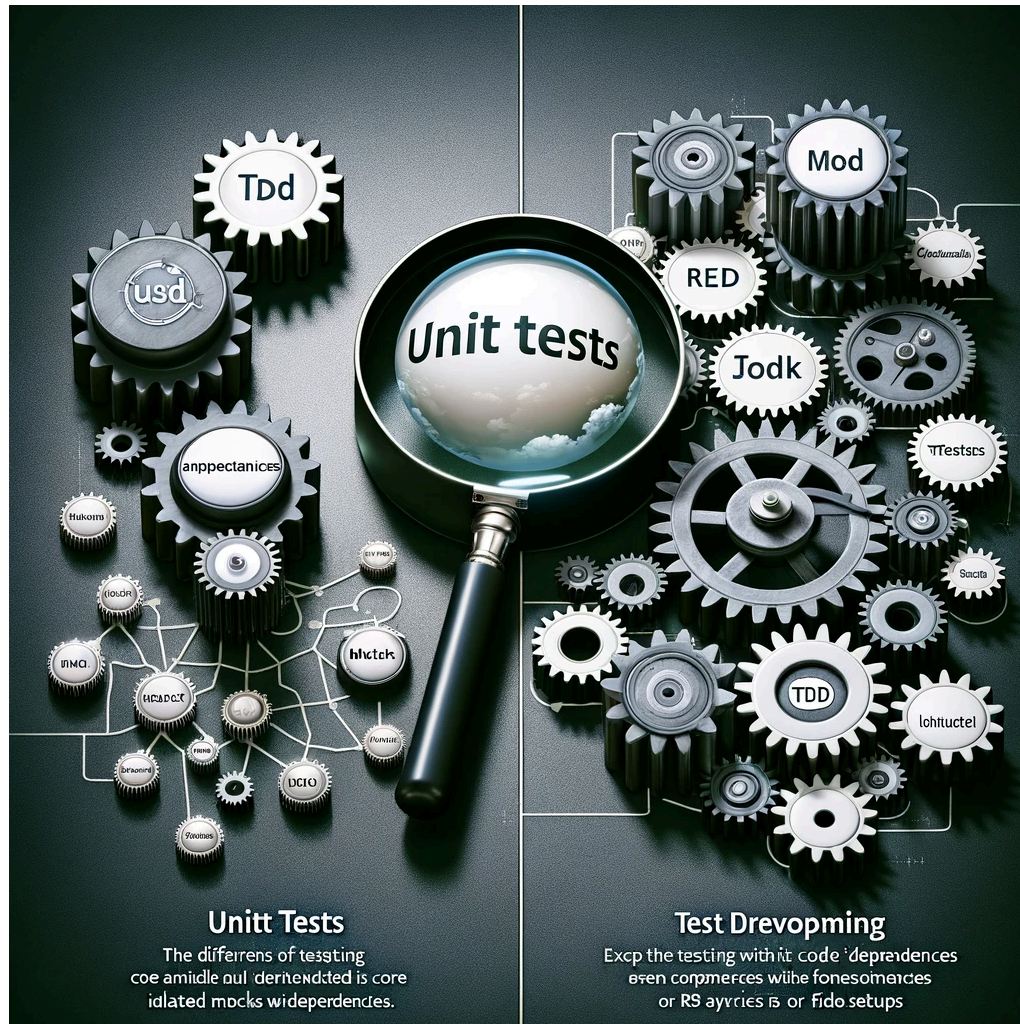
Irrtum 1: Softwareentwickler schreiben Unittest



Irrtum 1: Softwareentwickler schreiben Unittest

Falsches Verständnis von "Unit" .

- **alte** Wikipedia definition: "Unit testing refers to verifying the behavior of a **relatively small portion** of the code, a unit, in **isolation from the rest of the codebase**. A unit is often a **function**, module, **method** or **class**. [21] To isolate the unit, a test substitutes dependences with test doubles."
- **Wikipedia neu**: Unit testing, a.k.a. component or module testing, is a form of software testing by which isolated source code is tested to validate expected behavior.



Irrtum 1: TDD ≠ Unittesting

- Unit-Tests laut Wikipedia-Definition konzentriert sich auf die **Isolation** von Fehlern in einer **Unit** durch den **Einsatz von Test-Doubles**.

Irrtum 1: Das große "Unit" Missverständnis (1)

- Isolation der **Test**: JA!
- Isolation der getesteten "**Units**" durch "**test doubles**" : vermeide wenn möglich
- **Unit** = zusammengehörige Code-Teil (im Sinne von DDD) , also ein "Modul" mit definierter API : Prima !
- **Unit** = Java- oder TypeScript - **Klasse** : großes **Missverständnis** !

Irrtum 1: "Unit" Missverständnis (2)

- "Unit" im Sinne von Kent Beck (Autor von TDD): GUT!
- J-"Unit" als Tool: sehr hilfreich, (ebenso: Jasmin, Jest)
- "Unit" als einzelne Klasse und "wegmocken" aller Dependencies : **SCHLECHT!**

Irrtum 1: Warum sind "Unit" Tests schlecht? (1)

- zu enge Kopplung von Test zu Implementierung
- zu enge Kopplung an Interna (private methods) der Implementierung

Führt zu:

- Tests, die Refactoring extrem erschweren

Irrtum 1: Warum sind "Unit" Tests schlecht? (2)

Weil : Upfront-Design aller Implementierungsdetails nötig

- Test sollen das Design "treiben" ("driven") nicht umgekehrt
- Vorteil von TDD fürs Design wird verschenkt
- statt dessen : Upfront-Design **NUR** für die API gegen die die Test geschrieben werden

Korrektur des Irrtum 1 mit API (1):

- Teste Module mit stabiler API
- API-Design **vor** dem schreiben der Tests
- Implementierung **nach** dem schreiben des Tests
- Test greift nur auf stabile API zu
- bei Backends: **EINE** eine API für 5-20 Klassen



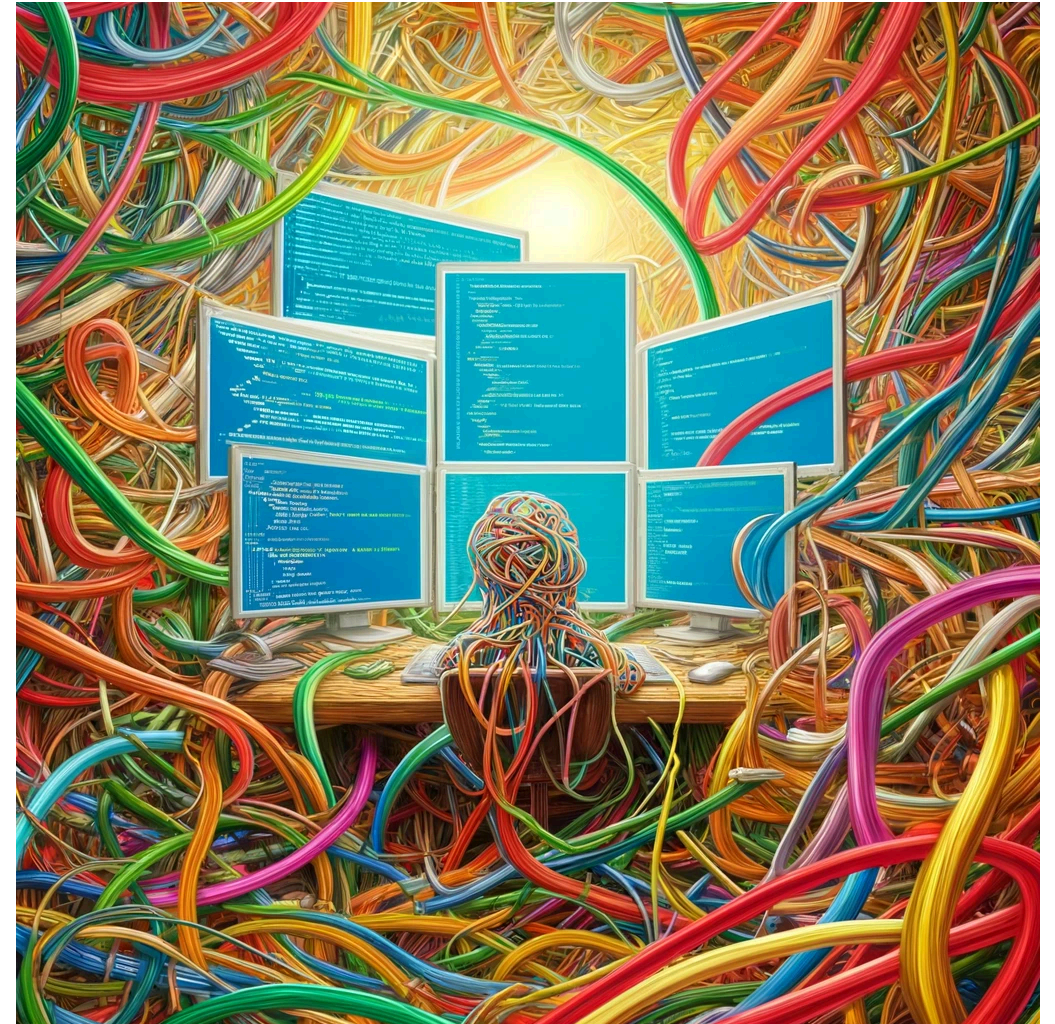
Irrtum 2: Verwende Mocks wo es nur geht

prüfe wo du Mocks oder andere Test Doubles verwendest

- Mocks für Klassen in DEINEM Code -> ganz schlecht
- viele Mocks -> enger Kopplung von Test zu Code -> deutlich erschwertes Refactoring

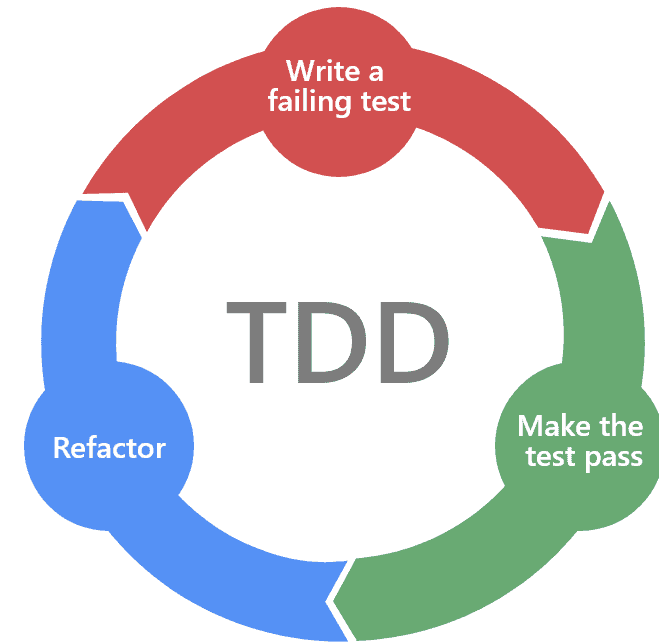
Irrtum 3: TDD ist unflexibel

- stimmt, wenn zu viel Mocks und Tests an die Implementierung statt an die Anforderungen gekoppelt sind
- stimmt, wenn du den "refaktorisieren" - Teil des TDD nicht wirklich ernst nimmst



Irrtum 3: TDD ist sehr flexibel wenn du

- größere "Units" mit API testtest
- wenig Mocks hast.
- sobald der Test grün 
●refaktorisieren●,





Irrtum 4: TDD kostet viel Zeit und verlangsamt die Entwicklung

- ja, wenn du es falsch angehst (siehe Irrtum 1 und 2)
- vielleicht ja: Projekt klein, kurzfristig oder nur Proof Of Konzept
- vielleicht ja: Legacy Projekt ohne gute Testabdeckung

Irrtum 4: TDD kostet viel Zeit und verlangsamt die Entwicklung

- nein, wenn du das Konzept wirklich ausnutzt
- nein, wenn dein Projekt groß und langfristig ist



Irrtum 5: TDD ist nur für erfahrene Entwickler

- nein: andere Denkweise 🤔 und Lösungsstrategie
- unabhängig von Vorerfahrung
- TDD **nicht** ein Ziel an dem man schnell ankommen will
- TDD ist ein **langer UND** sehr **interessanter** Weg



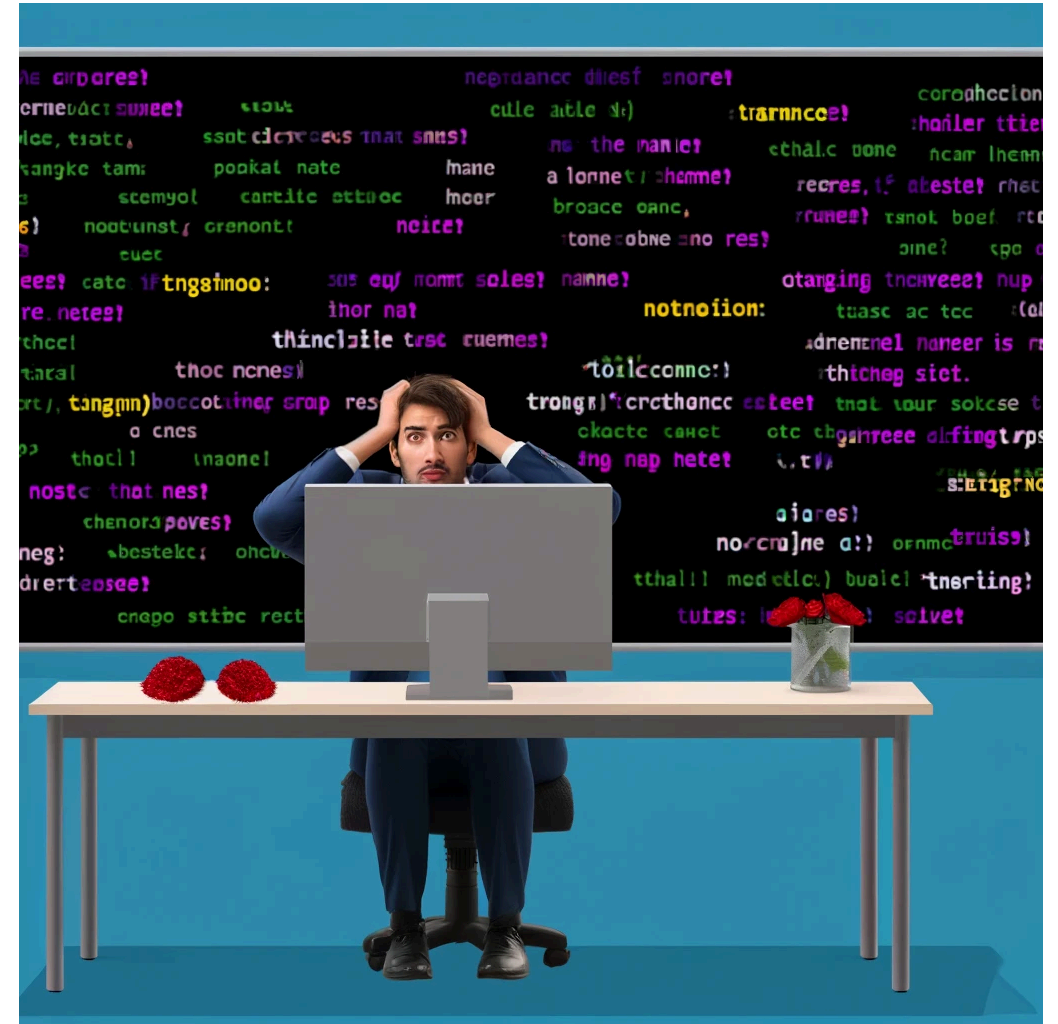


Irrtum 5: TDD ist nur für erfahrene Entwickler

- ⚡ Fehler:
 - Anforderung -> Implementierung :
FALSCH !! ⚡
- 😎 Statt dessen :
 - Anforderung -> Wie Anforderung testen ? 😎

Irrtum 6: die Namen meiner Test-Methoden sind egal

- wenn du das denkst, schießt du dir selber ins Knie,
- das Problem potenziert sich, wenn du auch noch viele Mocks verwendest

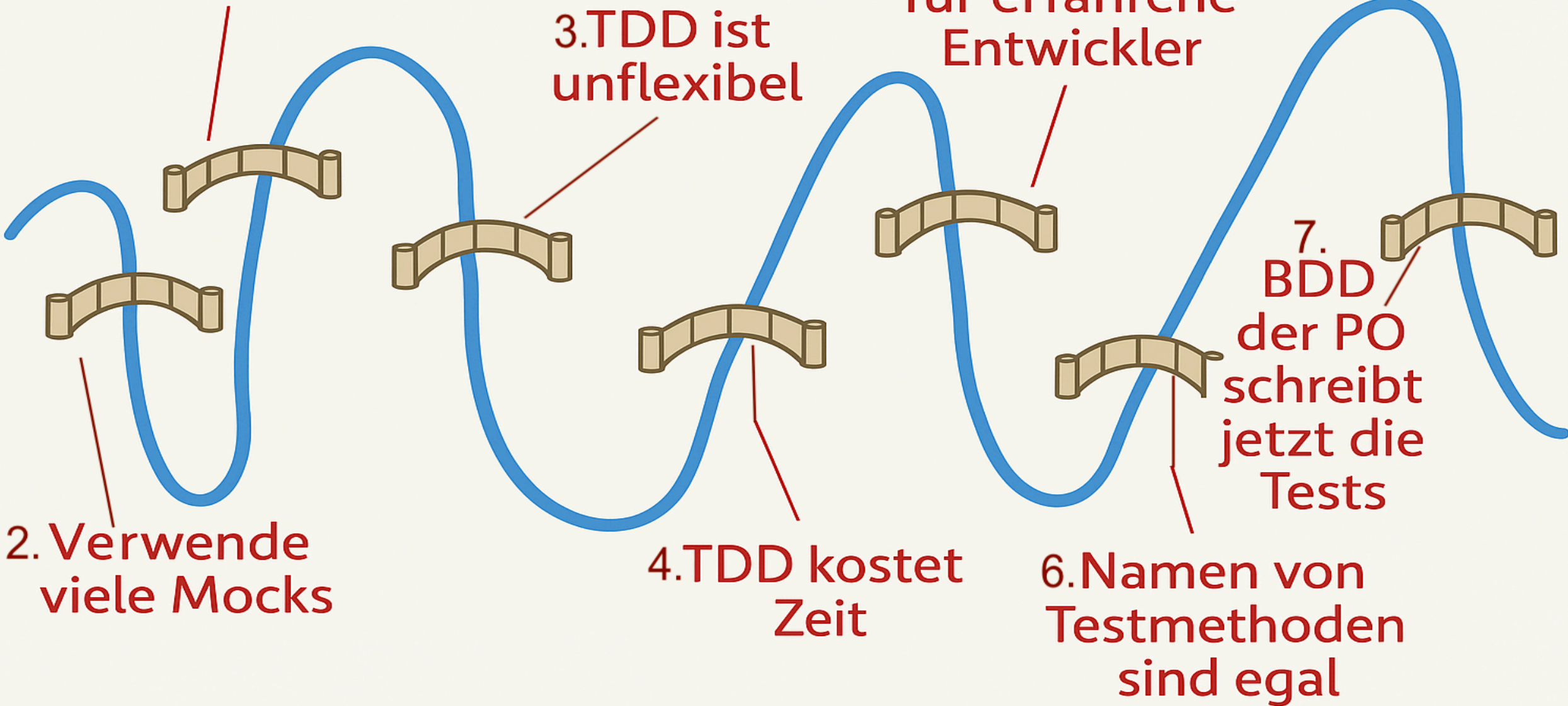


Irrtum 7: BDD ist toll: der PO schreibt jetzt meine Tests



- BDD = "Behaviour Driven Development"
- das wird nicht passieren, PO schreibt keine Tests
- BDD - tools wie Gherkin oder JBehave führen zu nichts als Overhead für den Entwickler
- besser : JGiven: Entwickler schreibt, PO liest

1. „Unit“ Testing



Nochmal "Über 7 Brücken musst du gehen, 7 dunkle Jahre überstehen" ?

▶ 0:00 / 0:28 — 🔊 ⋮

Aus Irrtum 6 + 7 kann was gutes entstehen (1):

- schreibe Tests-Methoden so, dass der PO den Test lesen, verstehen und gegen seine Anforderungen und Akzeptanzkriterien abgleichen kann
- Prima: jetzt verstehst du selber die Tests auch nach 4 Wochen noch!



Aus Irrtum $6 + 7$ kann was gutes entstehen (2):

einfaches Beispiel:

```
@Test
void test10() {
    driverPath =
        new File("/Users/bodo.teichmann/dev/learning/tddJavaMaven/chromedriver-mac-arm64/chromedriver");
    ChromeDriverService service = new ChromeDriverService.Builder()
        .usingDriverExecutable(driverPath)
        .build();
    WebDriver driver = new ChromeDriver(service);
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    try {
        driver.get("https://bodote.github.io/");
        String title = driver.getTitle();
        assertEquals("Die meisten Menschen... – Bodos Software Blog", title);
        WebElement searchLink =
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("fa-search")));
        searchLink.click();
        WebElement searchBox = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("search")));
        searchBox.sendKeys("TDD");
        WebElement post =
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.linkText("TDD Where it did go wrong")));
        post.click();
        WebElement postHeader = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("page-title")));
        assertEquals("TDD Where it did go wrong", postHeader.getText());
        List<WebElement> foundLinks = wait.until(
            ExpectedConditions.visibilityOfAllElementsLocatedBy(By.xpath("//li//ul/li/a | //li//ol/li/a")));
        var found = foundLinks.stream().anyMatch(link -> link.getText().contains("because"));
        assertTrue(found);
    } finally {
        driver.quit(); // Make sure to quit the driver to free up resources
    }
}
```

```
@Test
void should_find_a_blog_about_TDD() {
    given().a_web_site("https://bodote.github.io/");
    when().search_menu_is_clicked()
        .and()
        .search_term_$_is_typed("TDD");
    then().blog_post_$_should_be_found("TDD Where it did go wrong")
        .and()
        .when_clicked_$_should_be_the_title("TDD Where it did go wrong");
}
```


Der 2. Test erzeugt einen Report mit `jGiven`:

```
Test Class: jgiven.BlogTest
```

```
Should find a blog about TDD
```

```
Given a web site "https://bodote.github.io/"
```

```
When search menu is clicked
```

```
And search term "TDD" is typed
```

```
Then blog post "TDD Where it did go wrong" should be found
```

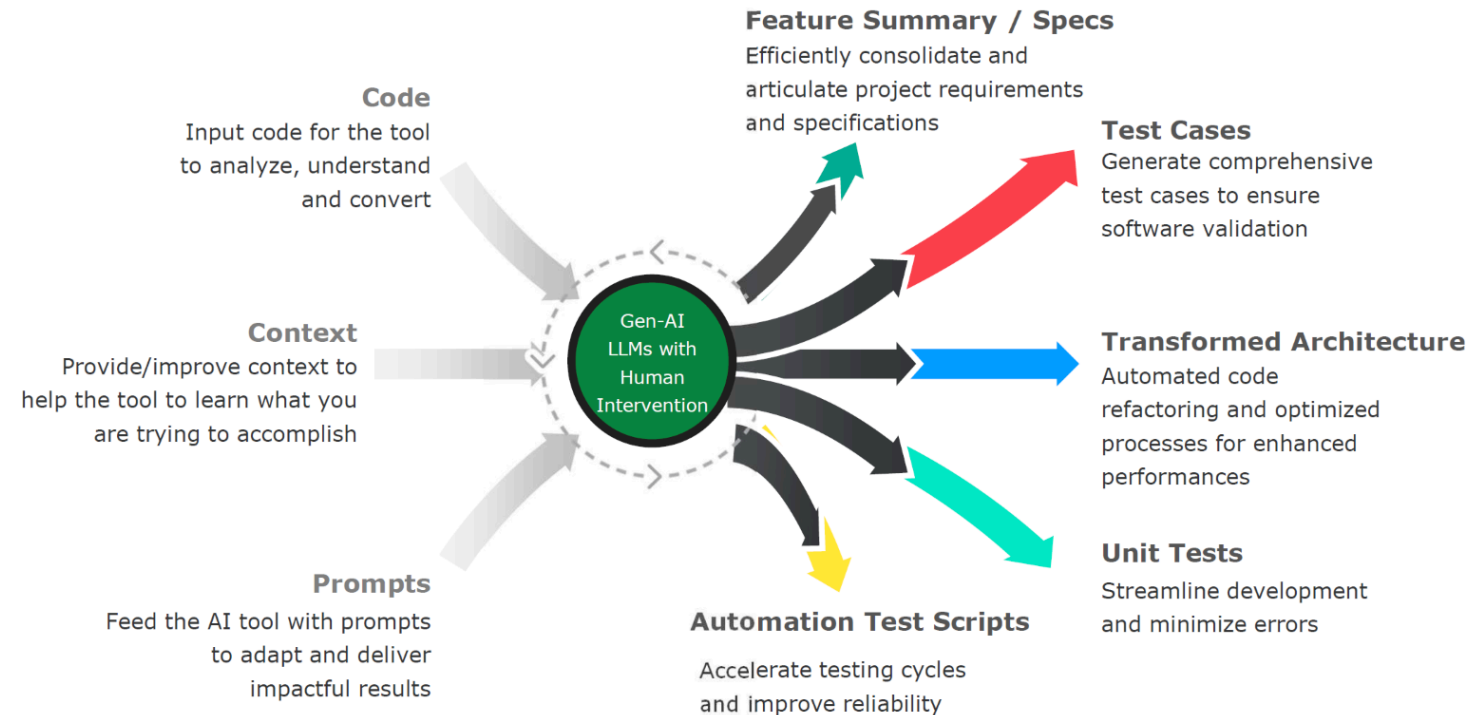
```
And when clicked "TDD Where it did go wrong" should be the title
```

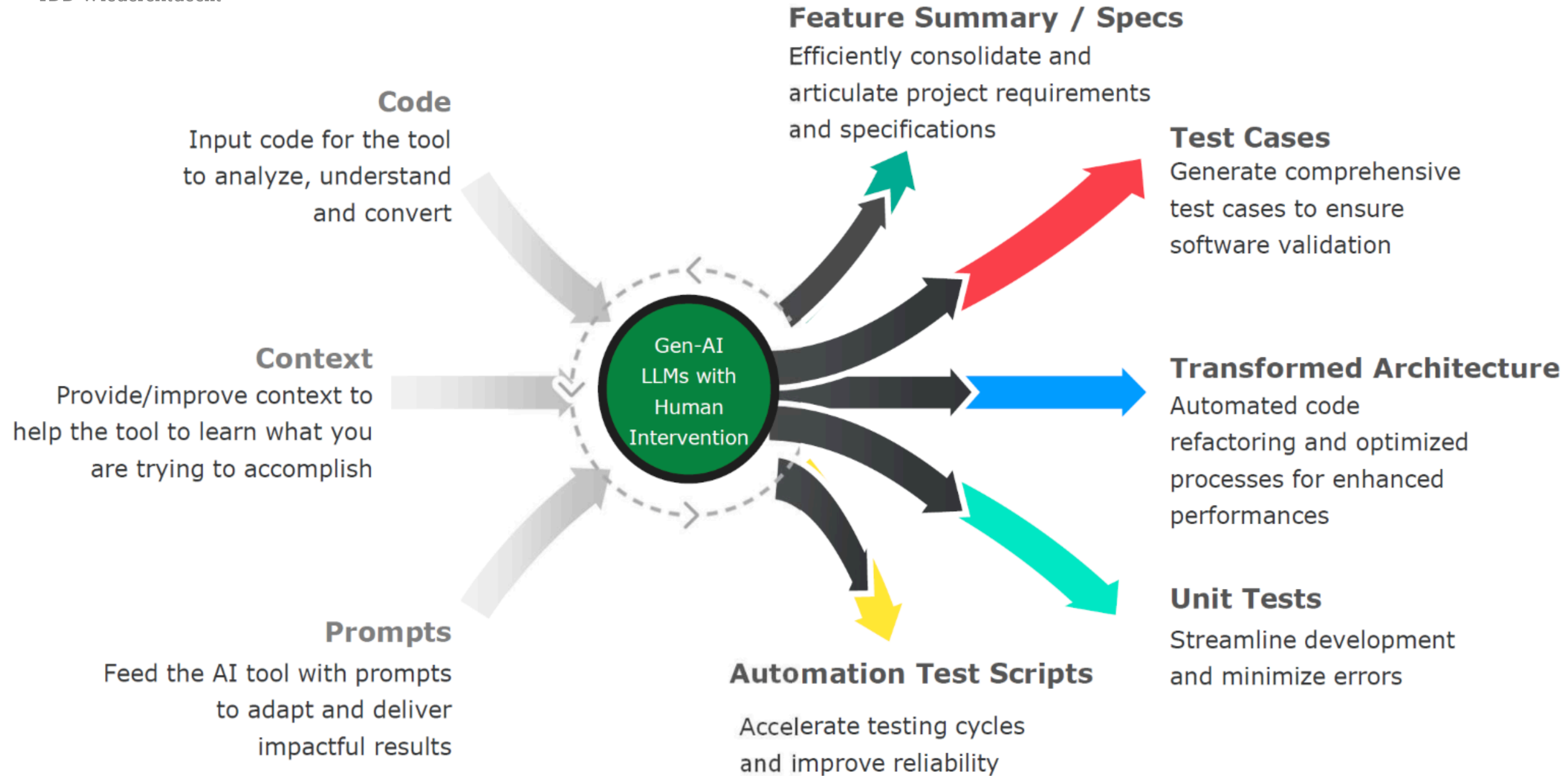
Sehr gut lesbare Testmethoden:

- helfen, sich schnell im Testcode zurecht zu finden, wenn sich Anforderungen ändern.
- dienen nun **tatsächlich** als Dokumentation

TDD und AI-assisted coding

Ist TDD in Zeiten von KI-unterstützter Softwareentwicklung noch sinnvoll ?





AI-assisted coding mit Cursor und TDD

Live Beispiel

Quellen:

- <https://bodote.github.io/blog/TDD-Video-Tips>
- praktisch alle Fehler schon selbst begangen, also eigene Erfahrung

Danke für eure Aufmerksamkeit!

BRANDAD Development GmbH

- Wir suchen Softwareentwickler, Scrummaster, Product Owner, UI/UX-ExpertInnen
- viel Angular und Spring-boot, aber auch anderen Frameworks, offen für neues
- <https://brandad.dev>

BRANDAD

Development

BRANDAD Development GmbH

- Wir suchen Softwareentwickler, Scrummaster, Product Owner, UI/UX-ExpertInnen
- viel Angular und Spring-boot, aber auch anderen Frameworks, offen für neues
- <https://brandad.dev>



Euer Feedback ist wertvoll!



Reservefolien

Welches Architekturmuster verwendest du ?

- keines
- layered
- hexagonal
- onion
- "Clean"

