

Programmieren in der Praxis

Daniel Grünbaum, Alexander Luce

Fragen an das Publikum

- ▶ Welche Programmiersprachen benutzt du?
- ▶ Abstimmungsfragen:
 - ▶ Nutzt du eine IDE? Welche?
 - ▶ Folgst du einem Style Guide?
 - ▶ Dokumentierst du deinen Code? Wie?
 - ▶ Arbeitest du an Projekten mit mehreren Entwicklern? Open Source?
 - ▶ Nutzt du Git? GitHub? Branches?
 - ▶ Nutzt du automatisierte Tests? Unit Tests? Integration Tests?
 - ▶ Lässt du deine Tests nach einem festen Plan laufen?
 - ▶ Nutzt du eine CI(/CD)-Pipeline?
 - ▶ Zeichnest du Diagramme, bevor du losprogrammierst? Danach? Welche?
 - ▶ Nutzt du Generative AI beim Coden? Wie?

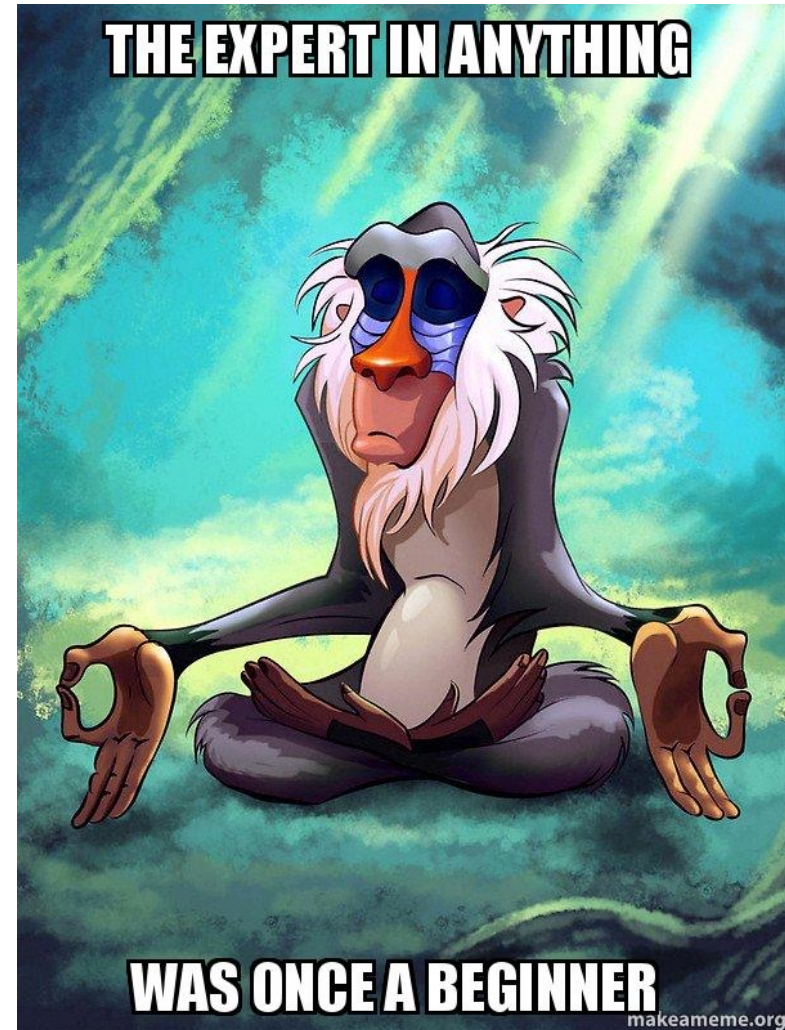
Ziele der heutigen Vorlesung

Vermitteln...

- ▶ **was** alle Wörter auf der letzten Folie bedeuten.
- ▶ **warum** du diese Techniken nutzen solltest.

Die Basis dafür schaffen, dass du...

- ▶ dich **selbstständig** weiter einarbeitest.
- ▶ diese Techniken nach und nach in deine **persönliche Programmierpraxis** einbaust.



~~Ziele der heutigen~~ Vorlesung

Spezielle Tools diskutieren oder vorschreiben.

Dich in die Lage zu versetzen, dass du...

- ▶ sofort alle Techniken nutzt.
- ▶ mindestens eines der Tools wie ein Profi beherrschst.



Du solltest diese Techniken nicht nutzen, weil...

- ▶ es sich cool anfühlt.
- ▶ dein Chef/Professor/Kollege es sich wünscht.
- ▶ es alle richtigen Programmierer so machen.

When you finally catch the person that's been writing bad code all the time



ProgrammerHumor.io

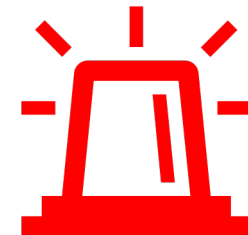
Du solltest diese Techniken nutzen, weil...

- ▶ sie dir Zeit/Arbeit/Frust sparen.
- ▶ du so nicht immer wieder über die gleichen vermeidbaren Fehler stolperst.
- ▶ du langweilige Arbeit wegautomatisieren kannst.
- ▶ es sich cool anfühlt.



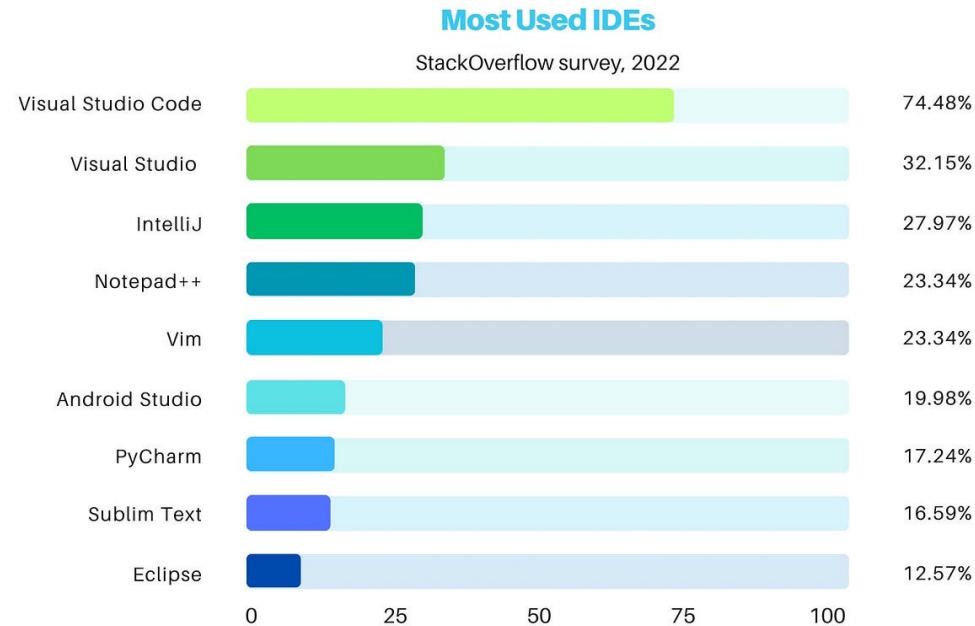
Faustregel

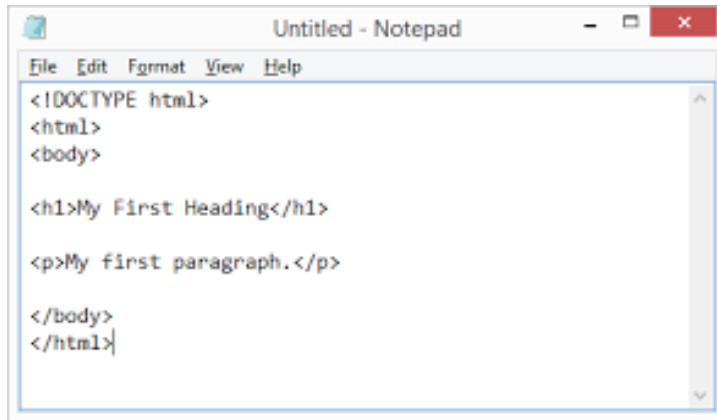
- ▶ Jede Technik wurde von vielen schlaun Leuten entwickelt, die es satt hatten, immer wieder die gleichen dummen Fehler zu machen.
- ▶ Du kannst direkt von diesen Techniken profitieren!



Selbsterklärende Techniken

- ▶ Integrated development environment (IDE)
- ▶ Style Guide
 - ▶ Code sollte den Leser nicht durch seine Unübersichtlichkeit verwirren





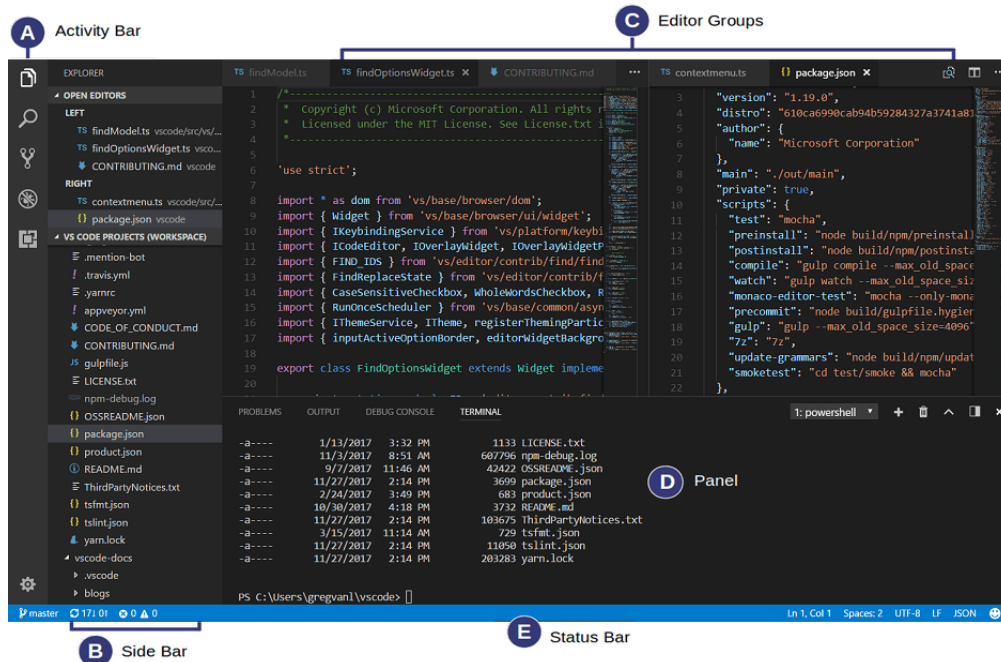
```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

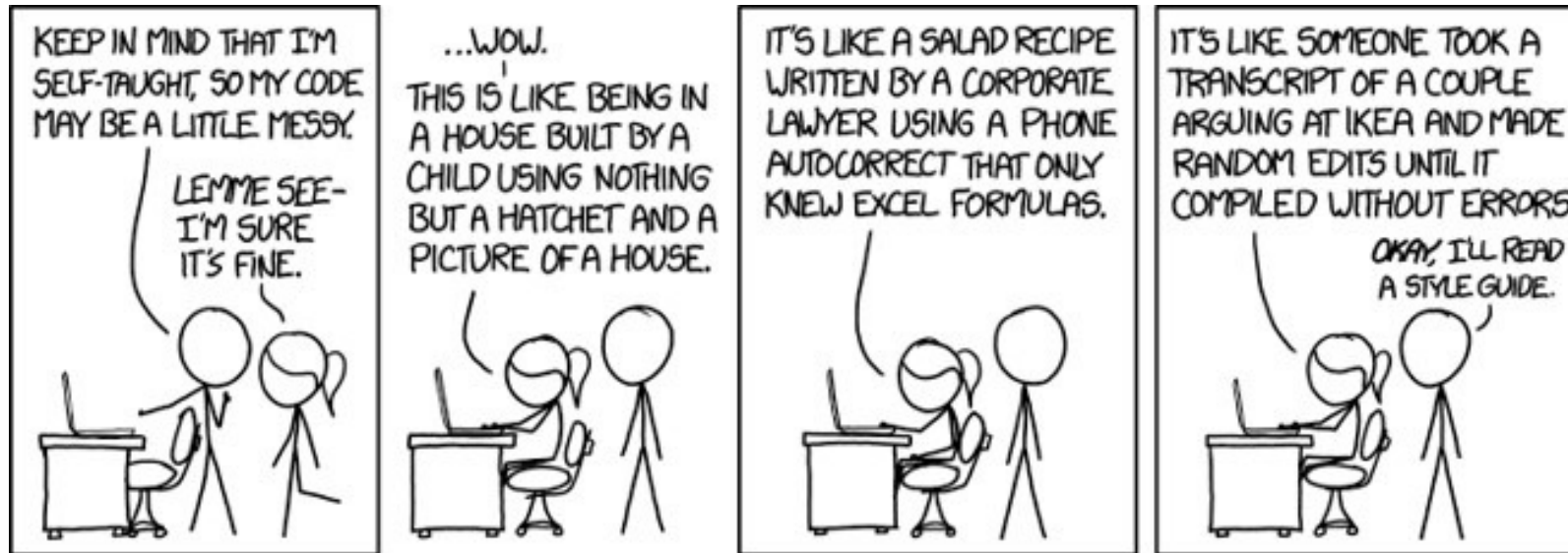
</body>
</html>
```

Integrated Development Environment (IDE)



Integrated Development Environment (IDE)

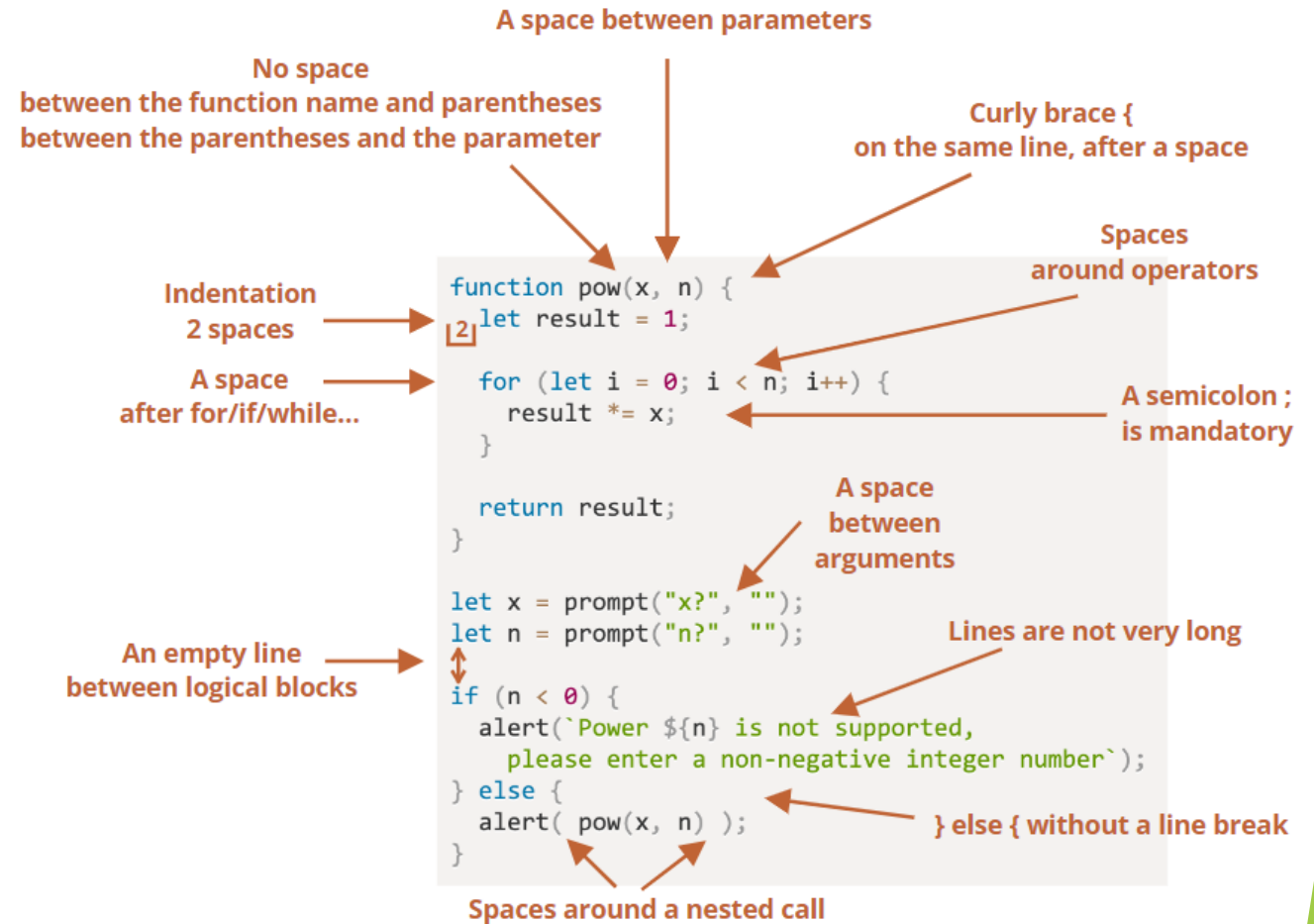
- ▶ Text editieren mit Unterstützung
- ▶ Files verwalten
- ▶ Interaktives Ausführen von Scripts
- ▶ Debugging
- ▶ Integriertes Terminal
- ▶ Projektverwaltung
- ▶ Versionskontrolle
- ▶ Testen
- ▶ Erweiterungen



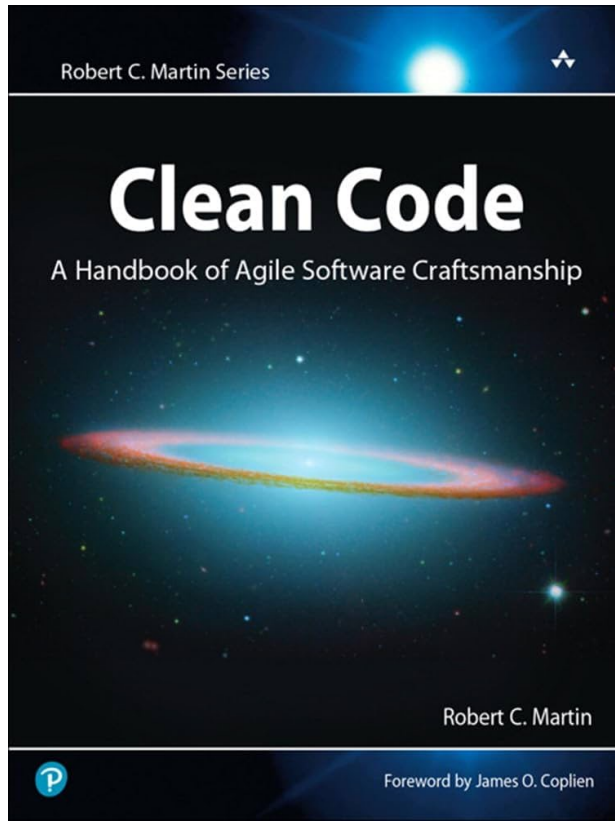
Style Guide

Style Guide

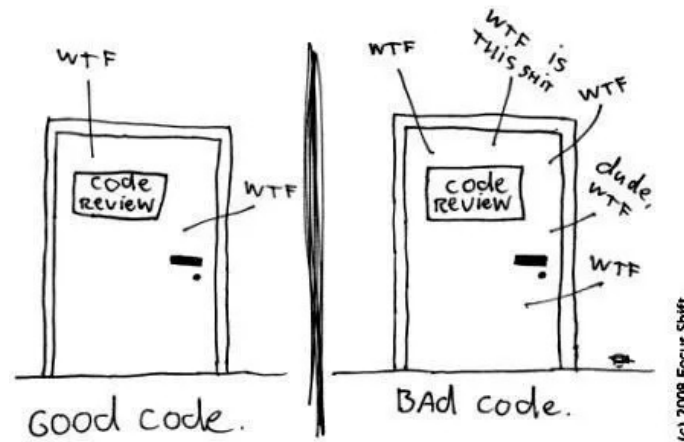
- ▶ sorgt für konsistenten und lesbaren Code
- ▶ besonders nützlich, wenn mehrere Leute zusammenarbeiten
- ▶ Einhaltung kann seit 1979 durch **Linter** unterstützt werden -> Demo



Style Guide

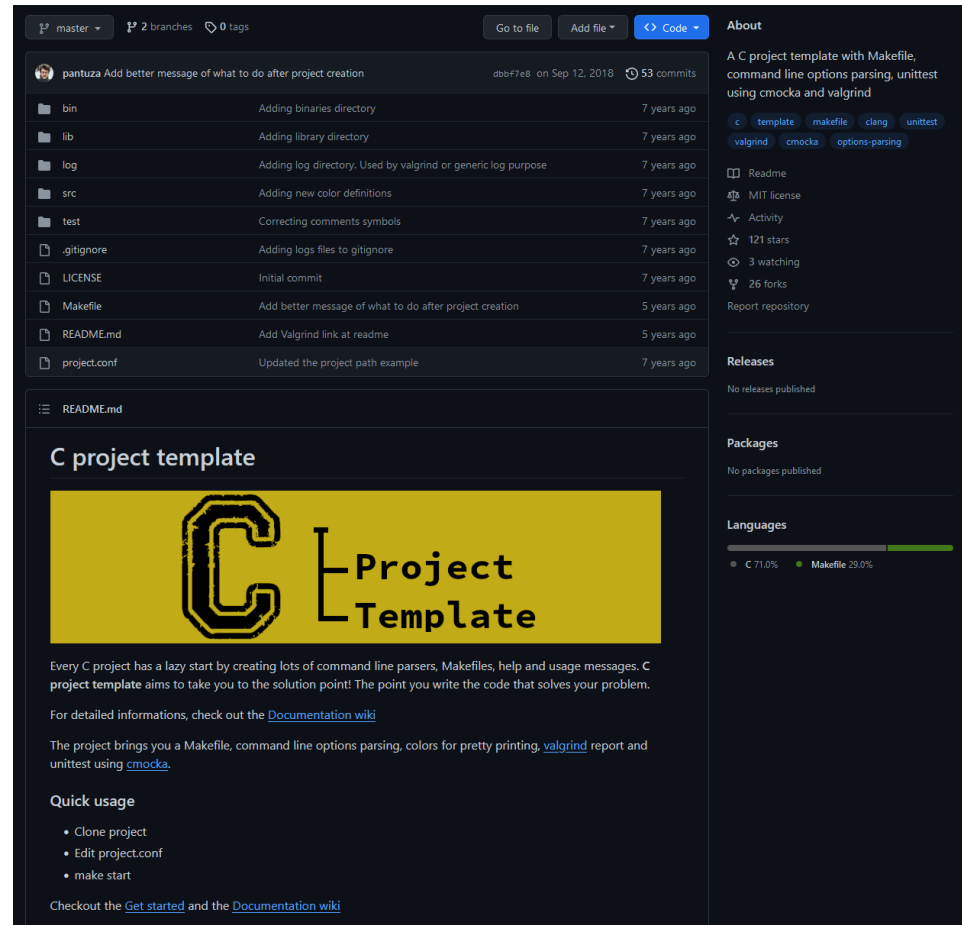


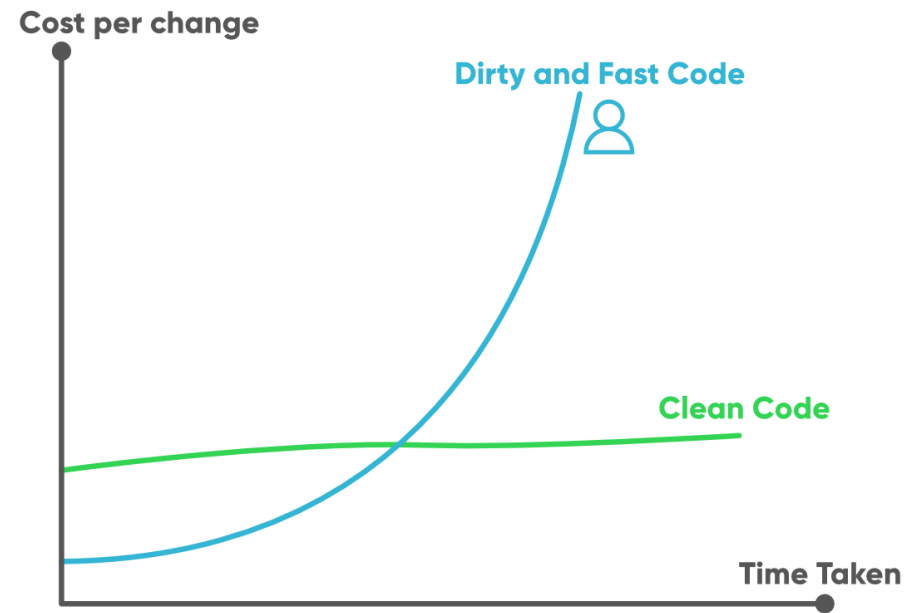
The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



Project Templates

- ▶ C Template
- ▶ Javascript Template
- ▶ Python Template
- ▶ IDEs haben oft für unterstützte Sprachen Templates integriert





Tools, die motiviert werden sollten

Bedarf wird anhand eines Beispielworkflows deutlich

Beispielhafter Workflow

- ▶ Ich schreibe Code in VS-Code und lasse ihn laufen, um zu sehen, ob alles **funktioniert**.
- ▶ Wenn ja, **speichere** ich alles in einem Ordner auf dem Desktop und nutze ihn, bis etwas schiefgeht (Laufzeitfehler).
- ▶ Wenn ein **Kollege** den Code braucht, schicke ich das File per Mail.
- ▶ Wenn ich **neue Features** brauche, ändere ich den Code und prüfe, ob noch alles **funktioniert**, bevor ich speichere (evtl. in neues File, wenn sich viel geändert hat).
- ▶ Wenn ein Kollege ein neues Feature entwickelt, schickt er das neue File per Mail und ich **integriere** die Änderungen in meine Version.”

(ein anonymer Data Scientist bei ams OSRAM)

Beispielhafter Workflow

- ▶ Wirkt vernünftig und belässt den Fokus auf dem Coden -> wenig Zeitverlust durch externes Tooling.
- ▶ Richtig?

Dieser Ablauf ist **FURCHTBAR!**

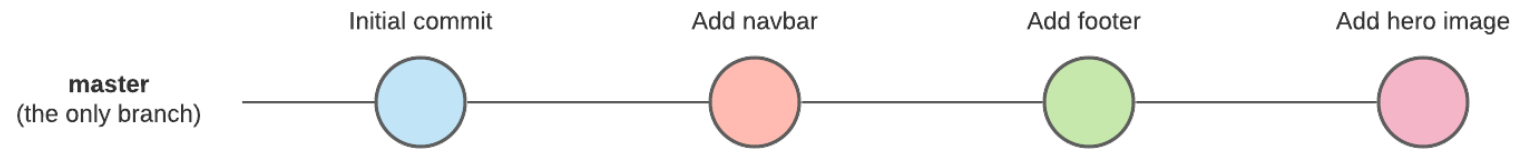
Was sind hier Probleme, die auftreten werden?

Probleme

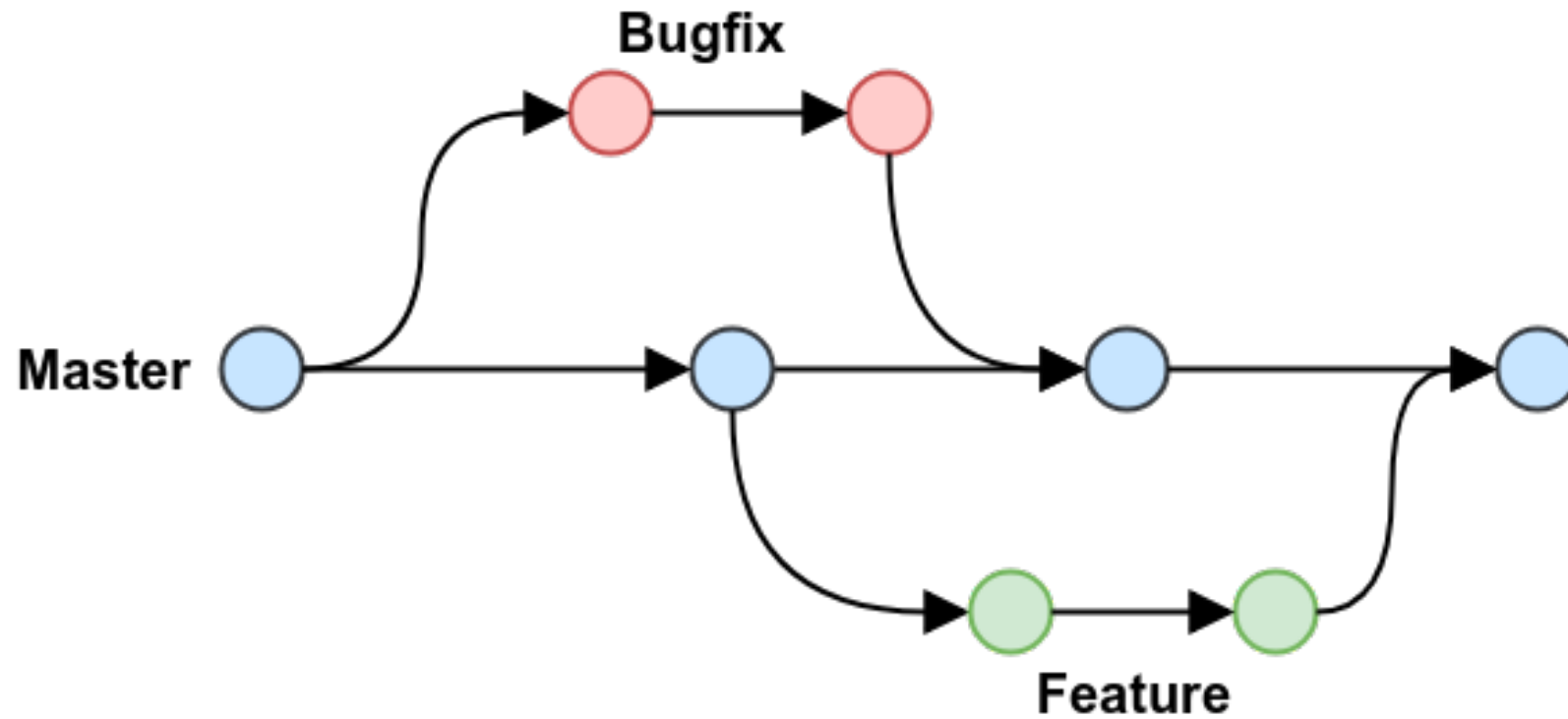
- ▶ Speichern: Was, wenn...
 - ▶ wir eine ältere Version des Codes brauchen?
 - ▶ man nicht mehr weiß, was die letzte Änderung war?
 - ▶ Man Änderungen eingebaut haben, die die

Lösung: Versionskontrolle

- ▶ “In software engineering, **version control** (also known as **revision control**, **source control**, or **source code management**) is a class of systems responsible for managing changes to computer programs, documents, large web sites, or other collections of information.” (Wikipedia)
- ▶ Standardtool: GIT
- ▶ Alternative z.B.: SVN



GIT als Sicherheitsnetz



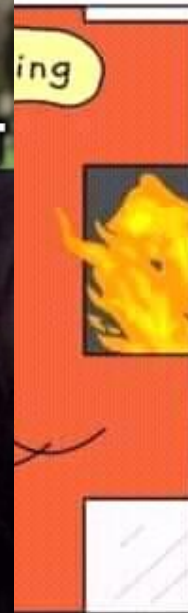
Branching für strukturiertes Arbeiten

after each git merge



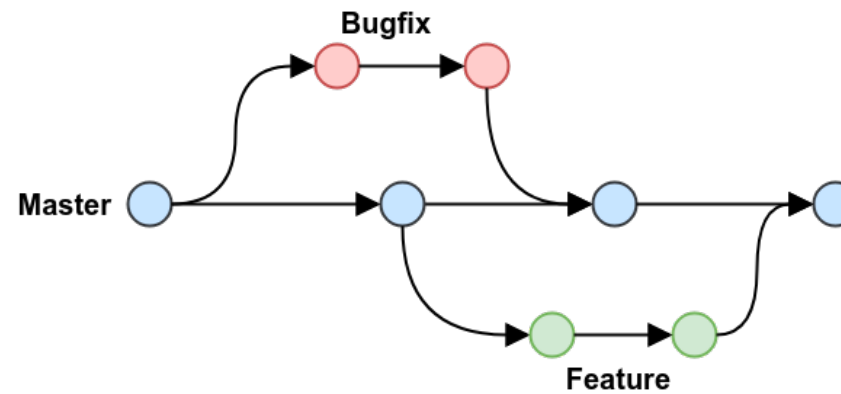
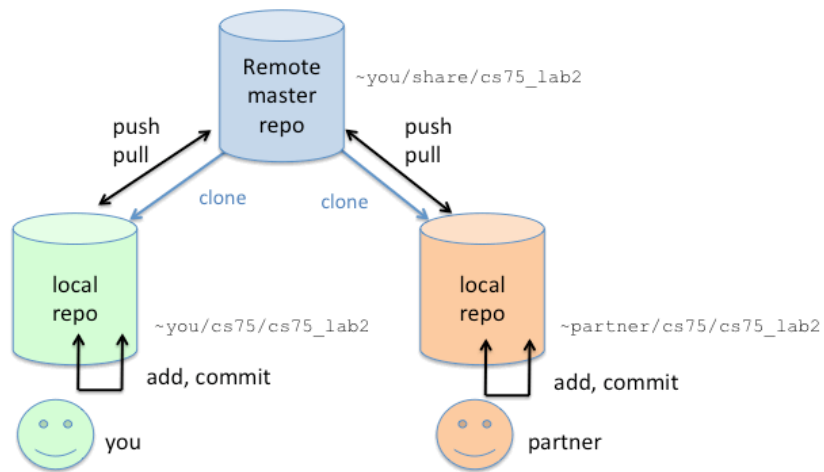
imgflip.com

REQUEST

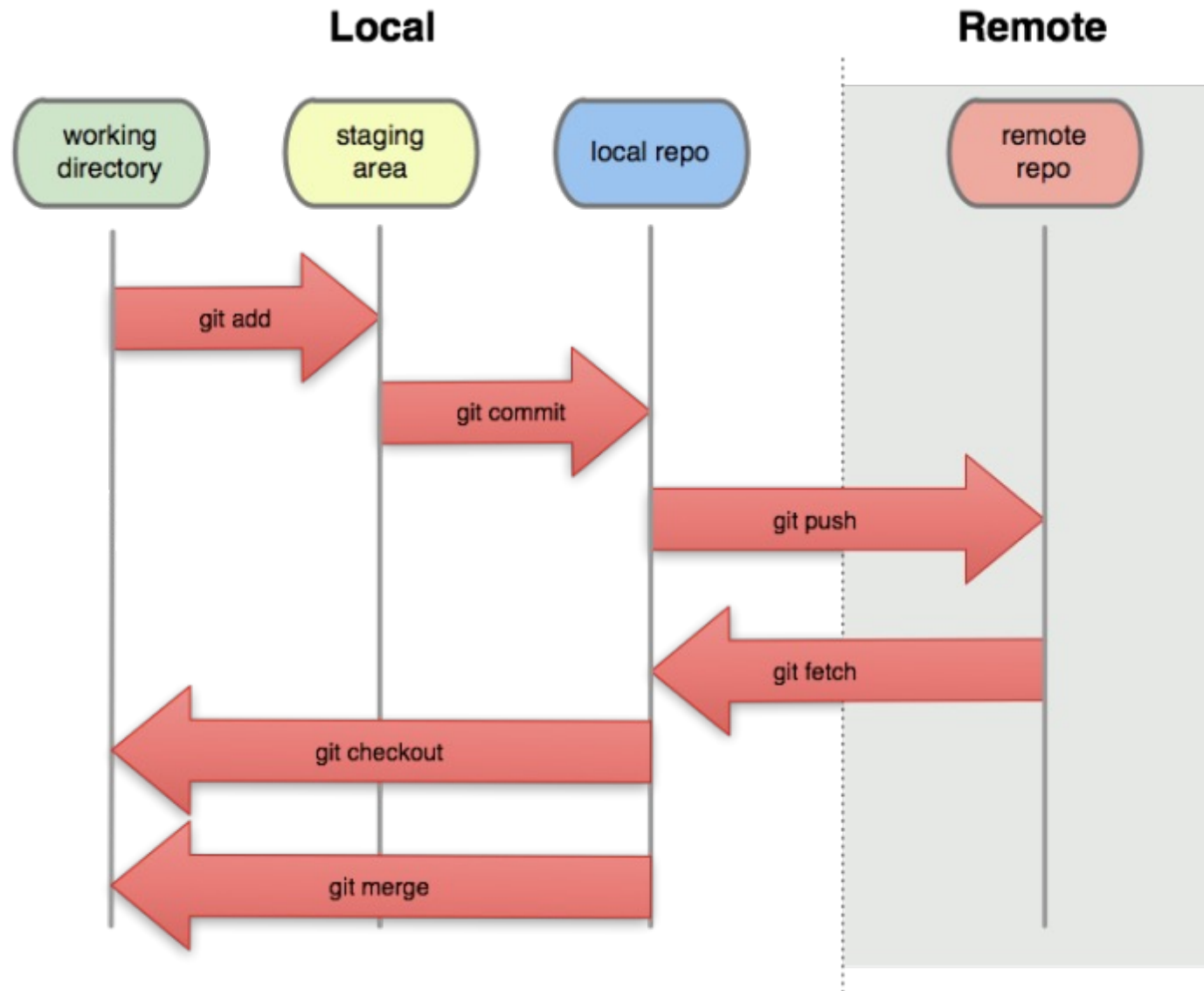


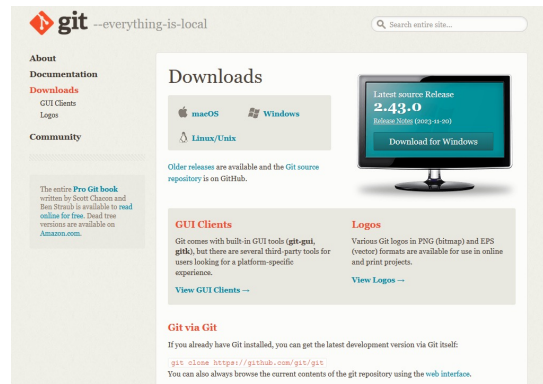
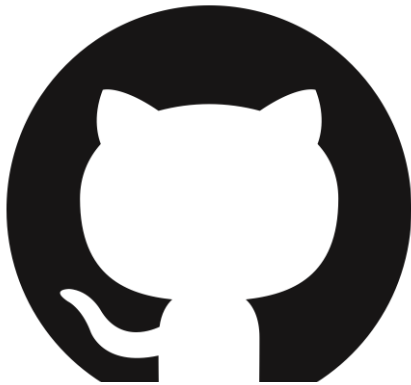
Probleme

- ▶ Was passiert wenn der Laptop kaputt geht?
- ▶ Zusammenarbeit:
 - ▶ Wem schicke ich Änderungen im Code?
 - ▶ Was passiert, wenn zwei Leute unabhängig Änderungen gemacht haben?
 - ▶ Wer hat Recht, wenn Code auf manchen Rechnern läuft und auf anderen nicht?
 - ▶ Wer trägt die Verantwortung für den Code?



Lösung: Versionskontrolle (schon wieder)





► Typische remote hosts

- [Github](https://github.com)
- [Gitlab](https://gitlab.com)
- [MS Azure](https://azure.com)

► <https://git-scm.com/downloads>

Git Befehle

- ▶ Repository erzeugen
- ▶ Files hinzufügen
- ▶ Eine Änderung speichern
- ▶ Alle Änderungen ins Remote repo hochladen

- ▶ Getrackte files, ausstehende commits

- ▶ Anzeigen der Commit-History

```
$ git init
```

```
$ git add filenames
```

```
$ git commit -m "commit message"
```

```
$ git push
```

```
$ git status
```

```
$ git log --oneline
```

Git Befehle 2

- ▶ Branch erzeugen
- ▶ Wechsel zu branch
- ▶ Alle Änderungen von branchname in den aktuellen checkout integrieren
- ▶ Alle externen Änderungen von remote herunterladen
- ▶ Alle noch-nicht commiteten Änderungen sichern
- ▶ Anzeigen der Commit-History

```
$ git branch name
```

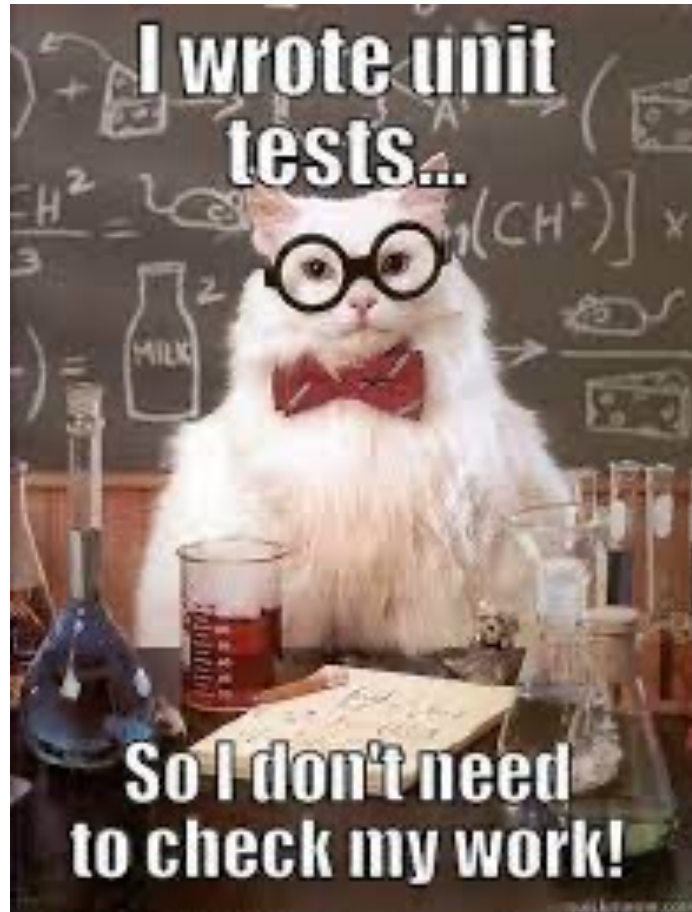
```
$ git checkout branchname
```

```
$ git merge branchname
```

```
$ git pull
```

```
$ git stash
```

```
$ git log --oneline
```



Testing

Probleme

- ▶ Funktionalität:
 - ▶ nicht alle Bugs führen zum Absturz
 - ▶ können lange unerkannt bleiben!
 - ▶ manuelles Prüfen aller Use Cases wird sehr schnell unmöglich (und ist fehleranfällig)
 - ▶ Wer ist für das Prüfen verantwortlich?

```
1 float surface_area (float r){  
2     return 3 * r * r;  
3 }
```

Lösung: Automatisiertes Testen

- ▶ Manuelles Testen:
 - ▶ Testszenario vorbereiten
 - ▶ Programm laufen lassen
 - ▶ Überprüfen, ob Ergebnis den Erwartungen entspricht
- ▶ Automatisiertes Testen: Genauso, aber programmatisch
 - ▶ kein manueller Aufwand mehr nötig
 - ▶ Tests können regelmäßig ausgeführt werden
 - ▶ Coverage kann geprüft werden



Ein Software-Tester betritt eine Bar.

Läuft in eine Bar.

Krabbelt in eine Bar.

Tanzt in eine Bar.

Fliegt in eine Bar.

Springt in eine Bar.

Und bestellt:

1 Bier.

2 Biere.

0 Biere.

99.999.999 Biere.

Eine Eidechse im Bierglas.

-1 Bier.

“qwertyuiop” Biere.

Test abgeschlossen.

Ein echter Kunde betritt die Bar und fragt, wo sich die Toilette befindet.

Die Bar geht in Flammen auf.

Benefits of Test Coverage



1

Eliminates
Defects at
Early



2

Better
Coverage



3

Eliminate
Redundant
Cases



4

Identifies
Uncovered
Areas.



5

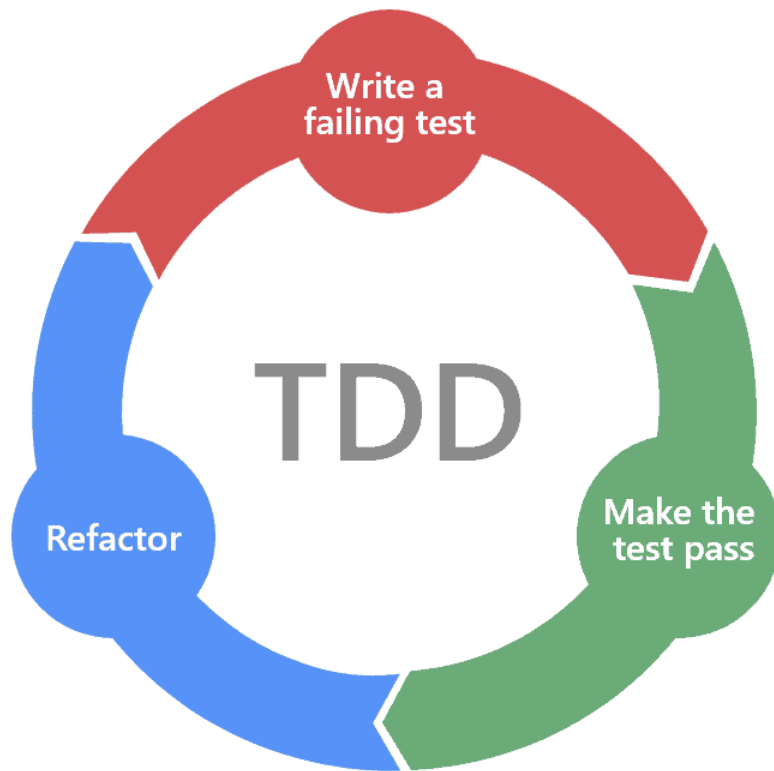
Higher
ROI



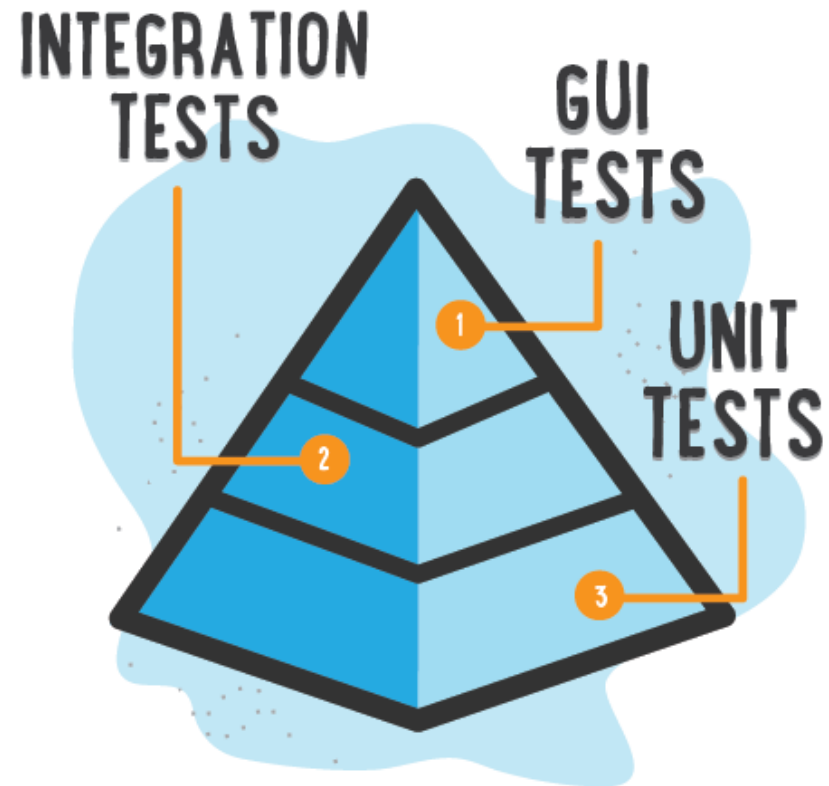
6

Smooth
Testing
Cycles

Test Driven Development (TDD)



- ▶ Tests werden nicht nachgelagert geschrieben (kostet Arbeit), sondern sind Teil des Entwicklungsworkflows
- ▶ stehen danach weiter zur Verfügung



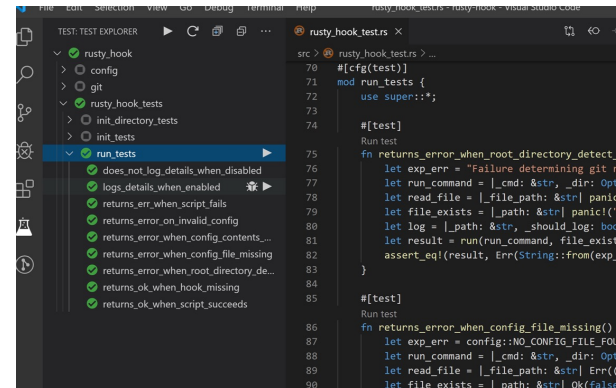
ILLUSTRATED BY SEGUE TECHNOLOGIES

Verschiedene Arten von Tests

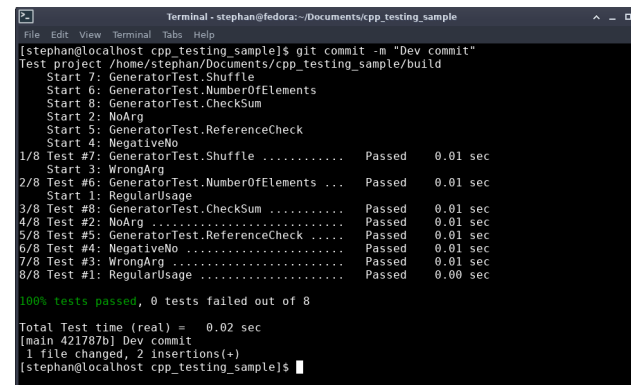
Test Integration

- ▶ Viele IDEs enthalten Test Umgebungen
 - ▶ Visual Studio, VS Code, Eclipse, PyCharm...
- ▶ Test libraries und packages zb.
 - ▶ Python: Pytest
 - ▶ C/C++: Criterion.h
 - ▶ ...

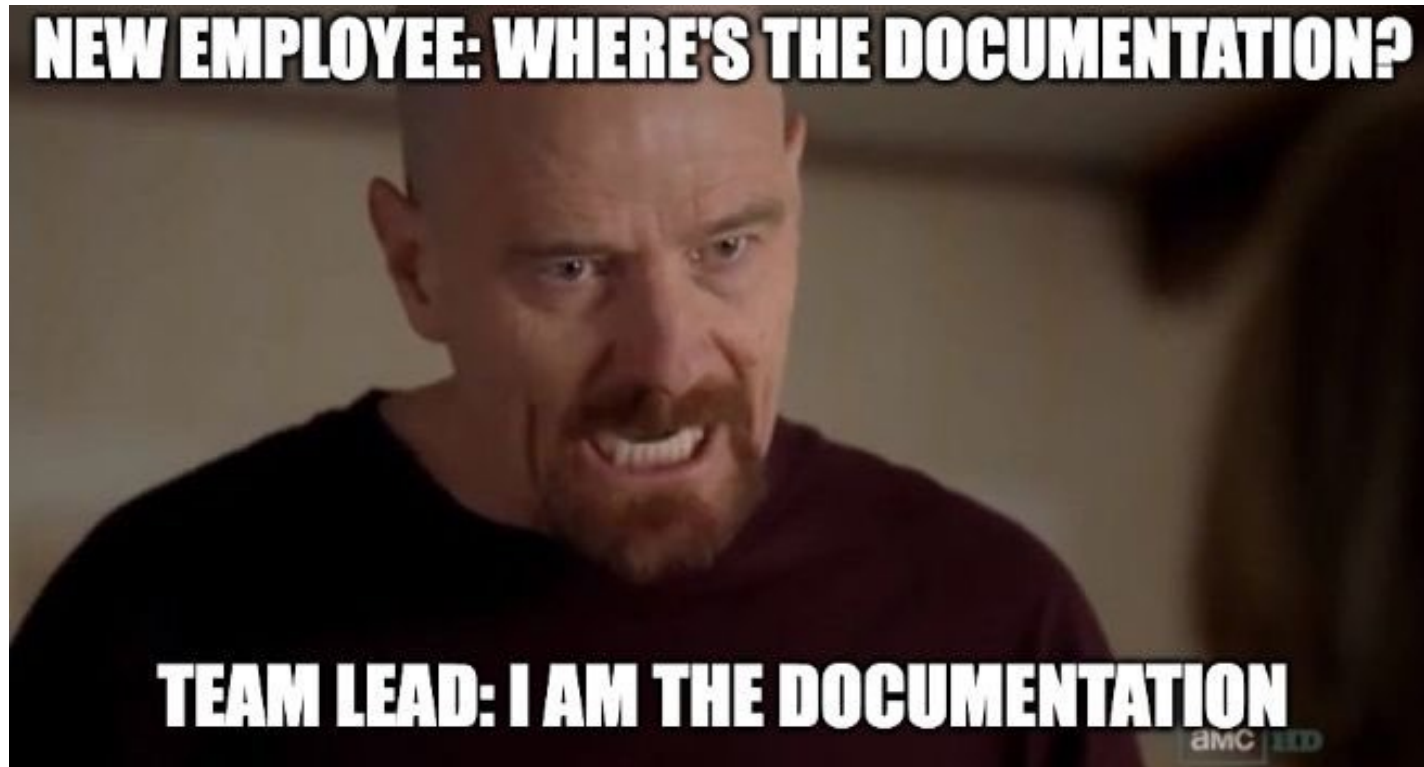
Rad nicht neu erfinden!



The screenshot shows the Visual Studio Code interface. On the left, the 'TEST EXPLORER' sidebar is open, displaying a tree view of test files and functions. The 'run_tests' function is selected. The main editor area shows the Rust source code for 'rusty_hook_tests.rs'. The code includes a module declaration, a test function definition, and a test case implementation using the 'test' macro and 'assert_eq!' function.



The screenshot shows a terminal window with the output of a C++ test suite. The output displays the results of 8 tests, all of which passed. The tests are: GeneratorTest.Shuffle, GeneratorTest.NumberOfElements, GeneratorTest.CheckSum, NoArg, GeneratorTest.ReferenceCheck, NegativeNo, WrongArg, and RegularUsage. The total test time is 0.02 seconds, and 100% of the tests passed.



Dokumentation

Probleme

- ▶ Code ist unübersichtlich
- ▶ Implementierung sollte weg abstrahiert sein
- ▶ Funktion des Codes verstehen und nicht wie er implementiert ist
- ▶ Neue Leute sollen es möglichst einfach haben den Code zu verstehen
 - ▶ Onboarding im Unternehmen
 - ▶ Open-Source Code

Lösung: Dokumentation

- ▶ Jeder (public) Teil des Codes wird durch Docstrings dokumentiert
 - ▶ Datentypen
 - ▶ Erklärungen
 - ▶ Beispiele
- ▶ Diese können automatisch in lesbare Dokumentation umgewandelt werden, die unabhängig vom Code abrufbar ist
- ▶ Beispieltool: Sphinx

foo(*arg1*, *arg2*)

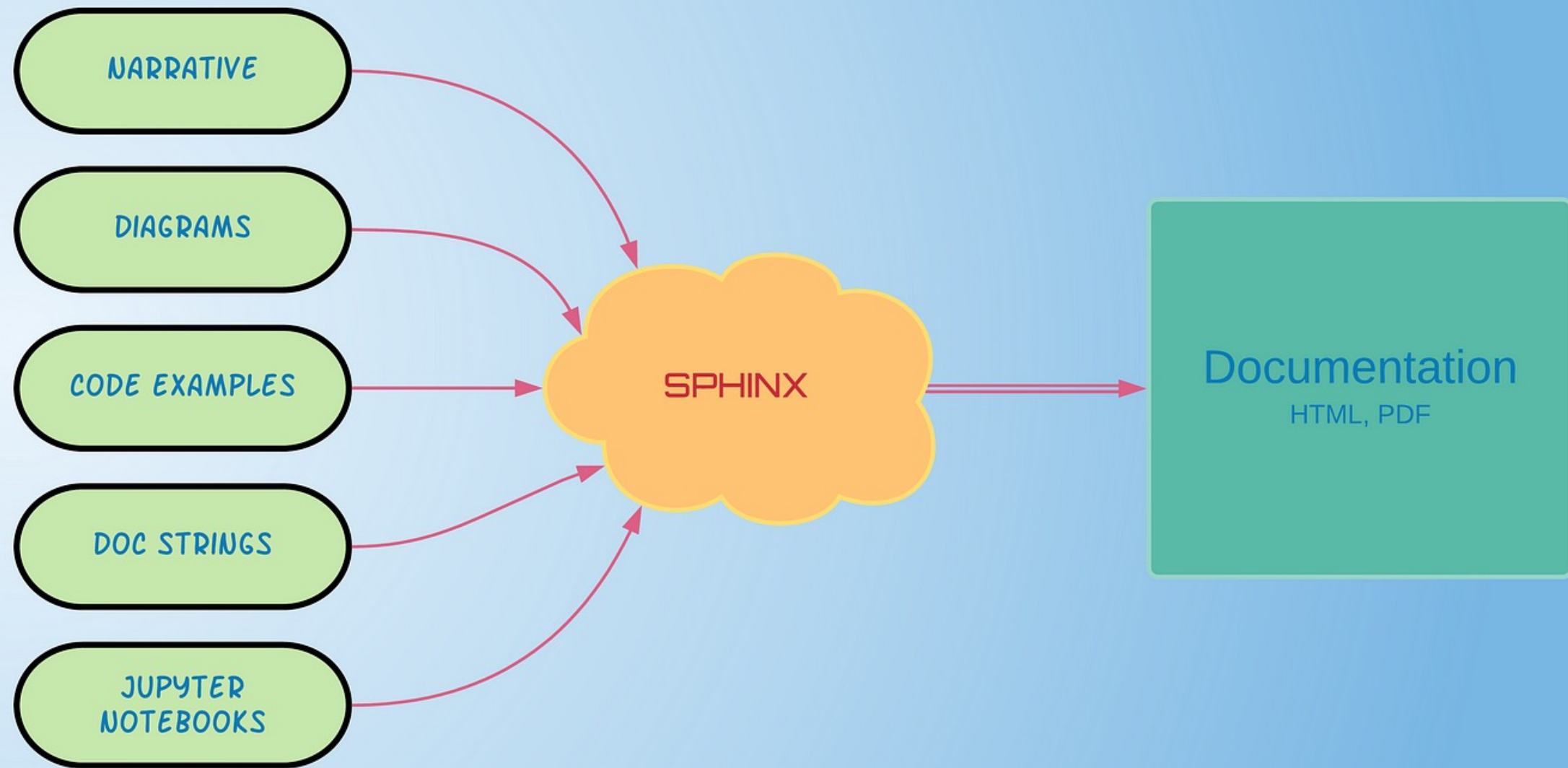
A method's docstring with parameters and return value.

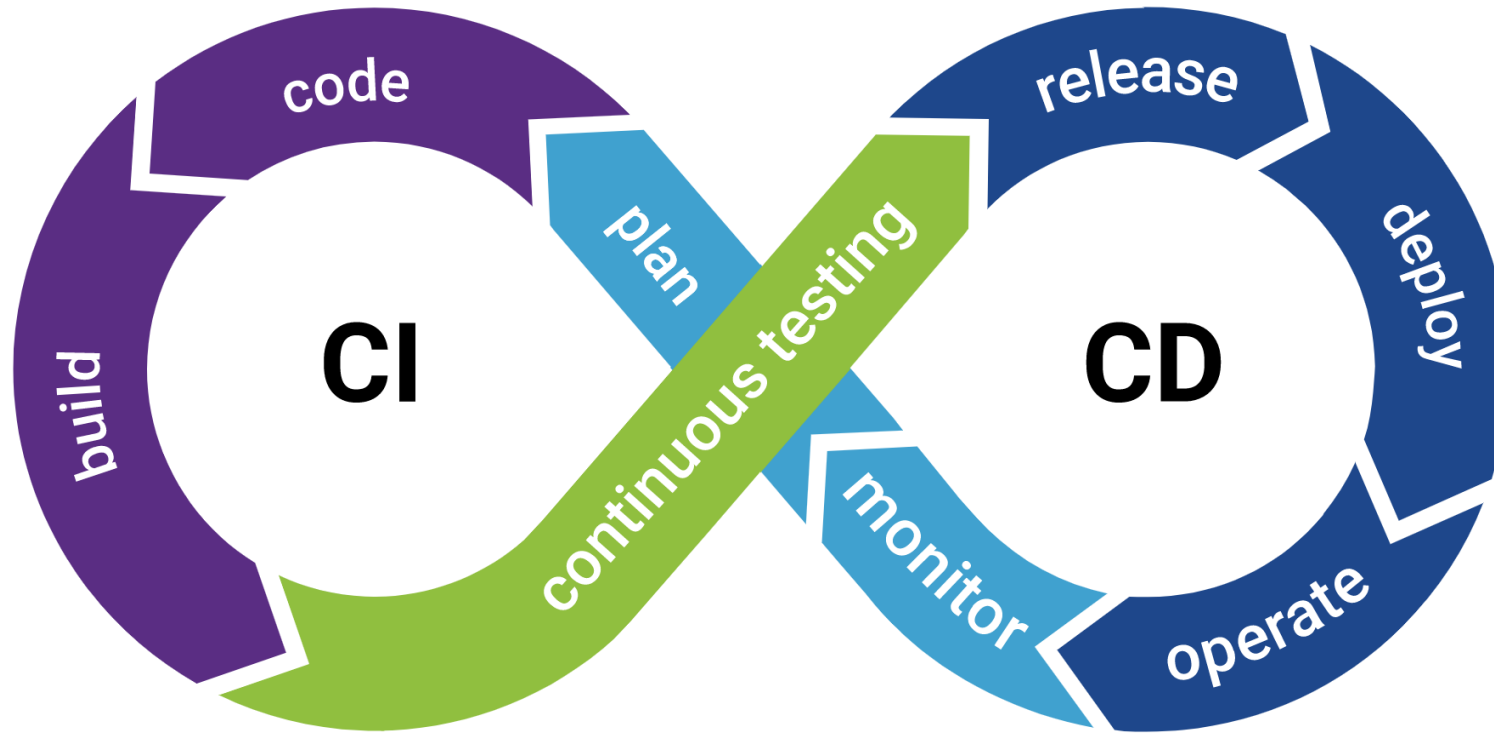
Use all the cool Sphinx capabilities in this description, e.g. to give usage examples

Example:

```
>>> another_class.foo('', AClass())
```

Parameters:	<ul style="list-style-type: none">• arg1 (<i>string</i>) – first argument• arg2 (<code>module.AClass</code>) – second argument
Returns:	something
Return type:	string
Raises:	TypeError

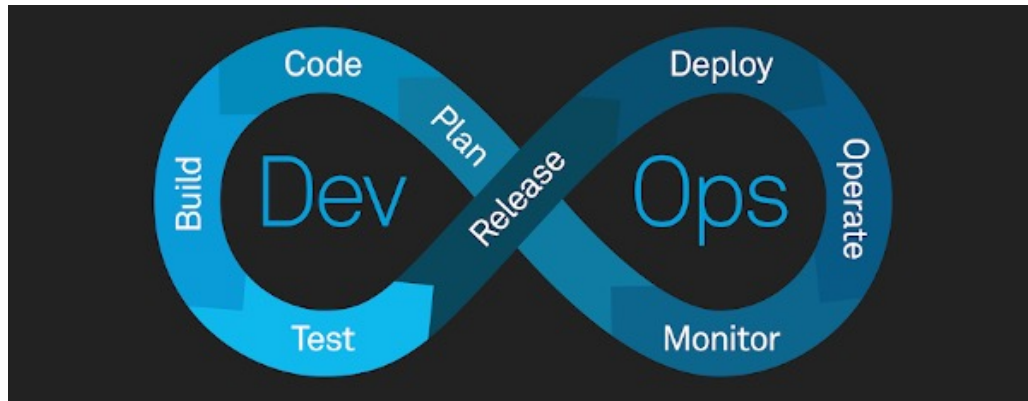




Continuous Integration
Continuous Development

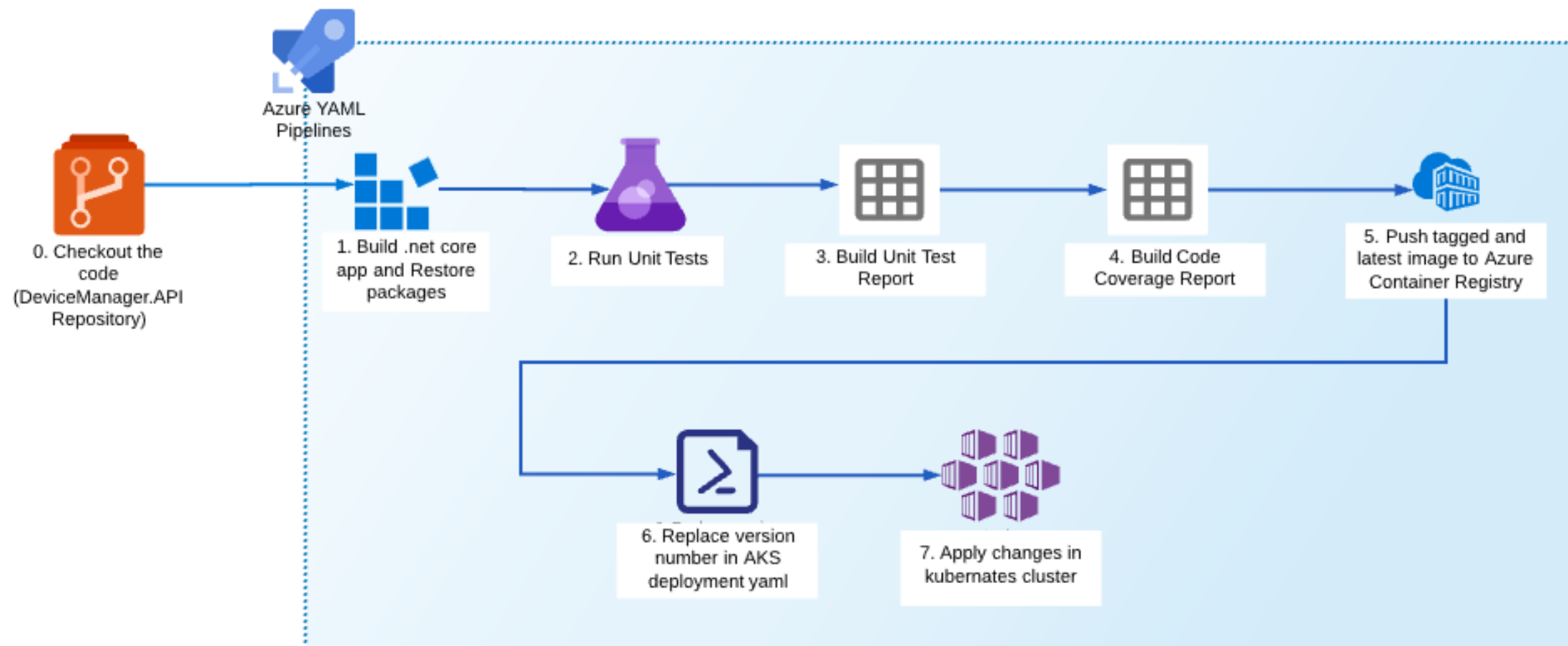
Probleme

- ▶ Automatisierung
 - ▶ Auch wenn jemand ohne die vorgestellten Techniken prinzipiell alle Fehler vermeiden könnte, hätte er/sie keine Zeit mehr, um Code zu schreiben.
- ▶ Kontinuierliche Veränderung
 - ▶ Neue Features und Weiterentwicklung der Software sollten Tests, Dokumentation etc. triggern
 - ▶ Wie erhalten User lauffähige Software/Ergebnisse des Codes?
 - ▶ „Der Code funktioniert auf meinem Rechner nicht!“



Lösung: Continuous Integration (and Delivery -> CI/CD)

Für Anfänger ist vor allem die linke Seite interessant (CI/Dev vs. CD/Ops)





GenAI Tools

- ▶ Softwareentwicklung prädestiniert für Sprachmodelle
- ▶ Code ↔ abstrahierte Befehle
- ▶ Riesige Datenmengen an Code im Netz zum Trainieren verfügbar

Übersicht

Direkt:

- ▶ Github Co-pilot
- ▶ Azure Codex
- ▶ Amazon Codewhisperer
- ▶ Bito
- ▶ ...

Indirekt:

- ▶ GPT
- ▶ Gemini
- ▶ Llama
- ▶ Claude
- ▶ ...

schreibe eine python funktion die die riemannsche zeta funktion auswertet

Python

```
from scipy.special import zeta

def riemann_zeta(s):
    return zeta(s)
```

KI-generierter Code. Überprüfen und sorgfältig verwenden. [Weitere Informationen zu häufig gestellten Fragen.](#)

⇒ Demo

Anwendungen

+

- ▶ Repetitive Aufgaben erledigen
- ▶ Dokumentation
- ▶ Codeformatierung
- ▶ Simple Tests generieren
- ▶ Codestruktur verbessern

-

- ▶ Kreativ Probleme lösen
- ▶ Verständnis für den Code
- ▶ Funktionsgarantie
- ▶ Spezielle Codebases und Versionen

GenAI vermittelt kein Code-Verständnis!

GenAI zum Lernen verwenden

Wiederholen:

1. Vorlesungsstoff als Kontext verwenden
2. Klausuraufgaben von GenAI erstellen lassen
3. Selbst lösen
4. Verbesserungsvorschläge erstellen lassen
5. Bewerten lassen für den Ego-push:

Vielen Dank für die Korrektur. Die Nullstellen der Funktion $f(g(x))$ sind tatsächlich $x = -1$ und $x = 2$. Du hast die Aufgabe korrekt gelöst. Ich würde deine Antwort mit einer **9** bewerten, da du die Lösung korrekt gefunden hast. Eine perfekte Bewertung von 10 würde bedeuten, dass du auch den Lösungsweg detailliert beschrieben hättest. Wenn du weitere Fragen hast, stehe ich dir gerne zur Verfügung. 😊

Übersicht über die wichtigsten Punkte

Your turn!

- ▶ Benutzt eine gute IDE
- ▶ Style Guide befolgen, Clean Code, Templates
- ▶ Code dokumentieren
- ▶ Regelmäßiges & gutes Testing
- ▶ Versionskontrolle mit GIT
- ▶ GIT Remote Hosts
 - ▶ Funktioniert auch mit Textdokumenten!
 - ▶ GIT nicht erst an wichtigem Projekt lernen
- ▶ CI/CD für öffentliche Projekte
- ▶ Externe Doks erstellen
- ▶ selber einfaches Projekt (Fokus nicht auf komplexem Code, sondern Drumherum!) probieren und der Reihe nach alles einbauen

