Testen

ARTEN VON TESTS

UNTERSCHEIDUNG

- *Größe* des Prüflings
- Aufwand für Testumgebung
- Anzahl der Anforderungen
  - Anzahl der Testfälle

DETAILS TESTEN

1. Unit Testing
   - *Modul* wird isoliert getestet
     - eine Klasse oder
     - eine Gruppe zusammenhängender Klassen
2. Integrations Testing
   - *Service-Test*
   - Zusammenspiel mehrerer Module
   - z.B. Datenbank & Importer-Modul

SMOKE TESTING

- alle wesentlichen Funktionen kurz ausprobieren
  - **keine** Detail-Tests
- Herkunft: Prüfen, dass das Gerät beim ersten Einschalten nicht brennt.

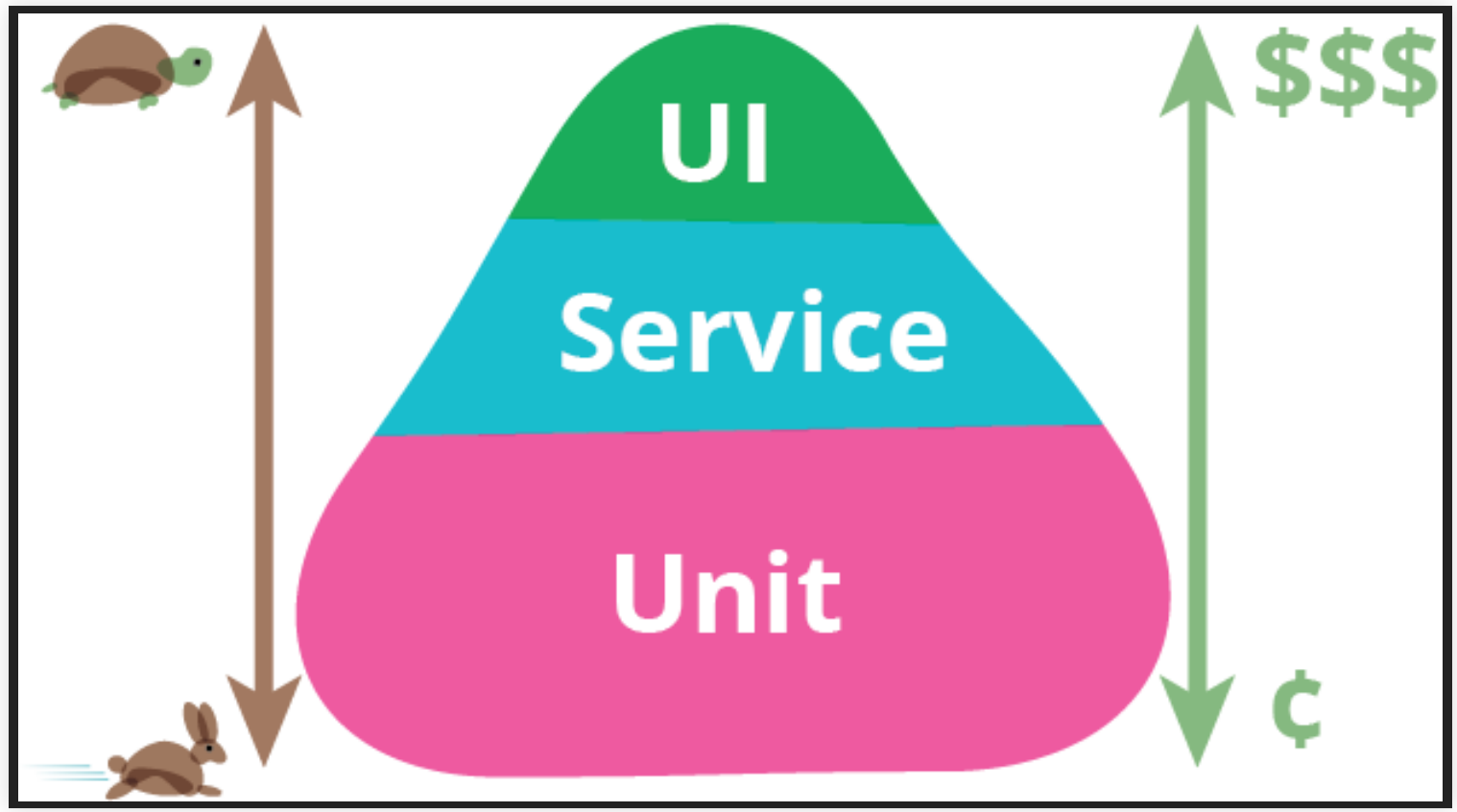
EXPLORATORY TESTING

- Tester lernt die Software beim Testen kennen
  - ad-hoc Entscheidungen, was getestet werden soll
- sinnvoll, wenn
  - keine/schlechte Spezifikation
  - wenig Zeit

END-TO-END TESTING

- Perspektive des Nutzers
- Zusammenspiel aller Module
- ggf. sehr umfangreiche Testumgebung
- Nachteile
  - langsam; spätes Feedback
  - Schwer, zu Pflegen
    - Folge: (meist) instabil
  - Schwer zu automatisieren
  - gefundene Fehler sind schwer zu lokalisieren

TESTPYRAMIDE



TESTTECHNIKEN

3.1

BLACK & WHITE

Wie kommen wir zu unseren Testfällen?

- Blackbox Tests
  - aus der Spezifikation/Anforderungen
- Whitebox Tests
  - durch Analyse der Codestruktur

TEST DRIVEN DEVELOPMENT

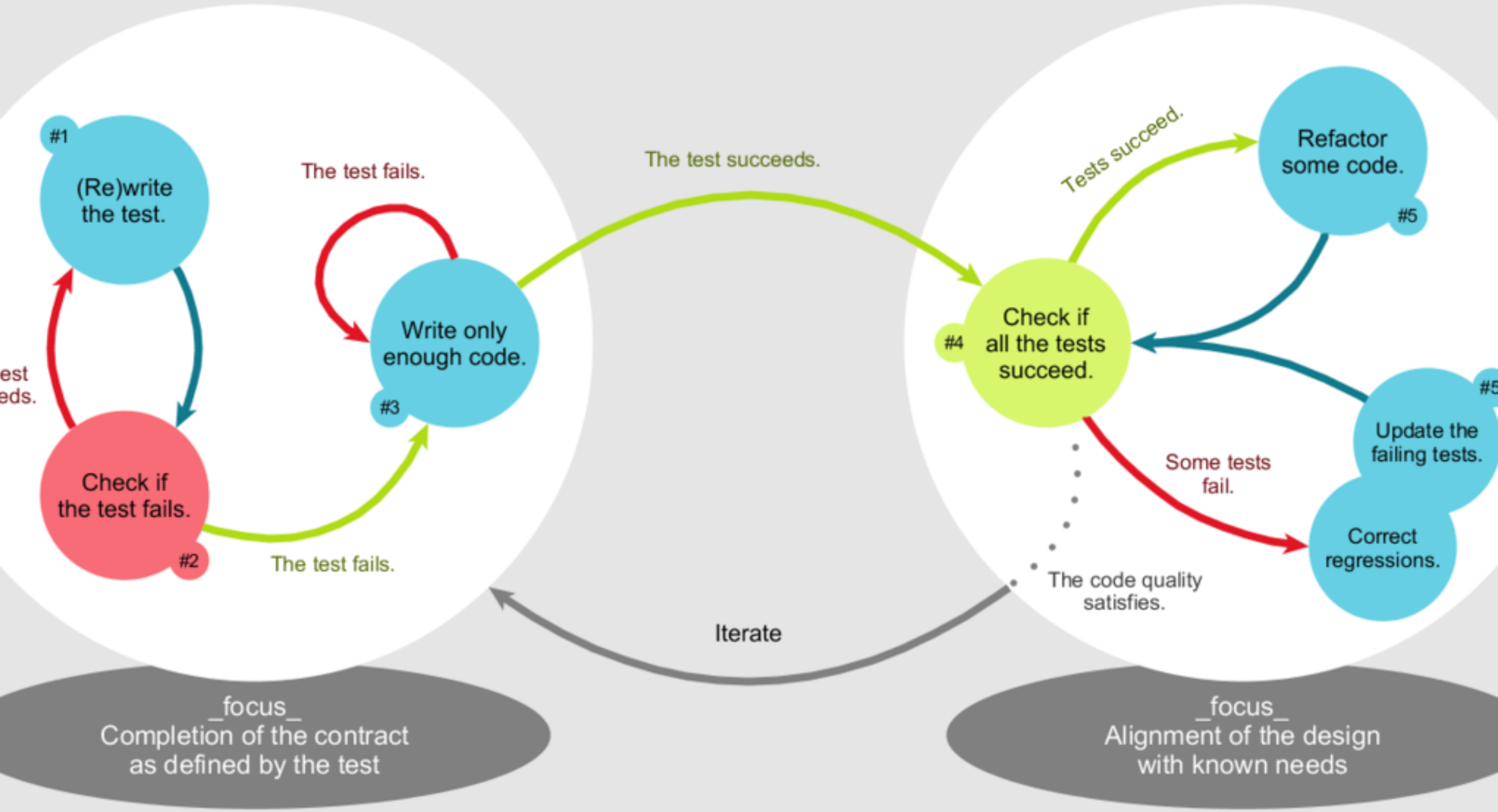
- Test First
- Anforderungen werden sofort/zuerst in Testfällen ausgedrückt
- es muss nur der Code geschrieben werden, der nötig ist um die Tests zu bestehen

KISS

Keep it simple, stupid

YAGNI

You aren’t gonna need it


TEST-FIRST DEVELOPMENT
REFACTORING
#1
(Re)write the test.
The test fails.
Write only enough code.
#3
The test succeeds.
Check if the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
Check if all the tests succeed.
#4
Some tests fail.
Update the failing tests.
Correct regressions.
The code quality satisfies.
Iterate
focus
Completion of the contract as defined by the test
focus
Alignment of the design with known needs

TEST DRIVEN DEVELOPMENT

- Vorteile
  1. TestCode beschreibt den getesteten Code selbst
  2. fördert kleine Module
  3. fördert testbare Software
  4. Code-Design wird modularer, flexibler
- Nachteile
  1. Blinde Flecken werden vom Coder & Tester nicht gesehen (gleiche Person)
  2. spätere Änderungen an Architektur sind aufwendig
  3. Testcode muss auch gewartet werden

TESTABDECKUNG

Wie viele Testfälle müssen geschrieben werden?

C0

Durchlauf jeder Anweisung

C1

Durchlauf jedes Zweiges, auch der leeren

```
int z = x;
if (y > x) {
   z = y;
}
z = z * 2;
```

TESTABDECKUNG

Wie viele Testfälle müssen geschrieben werden?

C2

Durchlauf aller möglichen Pfade; Schwierig bei Schleifen

```
if (y > x) {
   z = y;
} else {
   z = x;
}
if (x == 2 | y == 2 ) {
   z = z * 2;
} else {
   z = z * 4;
}
```

TESTABDECKUNG

Wie viele Testfälle müssen geschrieben werden?

C3

Durchlauf mit allen möglichen Bedingungen

C3a

Jede atomare Bedingung einer Entscheidung muss einmal mit true und einmal mit false getestet werden.

C3b

Alle Kombinationen der atomare Bedingung einer Entscheidung müssen getestet werden.

TESTABDECKUNG

Wie viele Testfälle müssen geschrieben werden?

C3a

Jede atomare Bedingung einer Entscheidung muss einmal mit true und einmal mit false getestet werden.

```
if (x == 2 | y == 2 ) {
   z = z * 2;
} else {
   z = z * 4;
}
```

- zwei Testfälle x,y: 1,1 2,2

TESTABDECKUNG

Wie viele Testfälle müssen geschrieben werden?

C3b

Alle Kombinationen der atomare Bedingung einer Entscheidung müssen getestet werden.

```
if (x == 2 | y == 2 ) {
   z = z * 2;
} else {
   z = z * 4;
}
```

- vier Testfälle x,y: 1,2 3,2 3,1 2,2

TESTABDECKUNG

- 100% Coverage kein gutes Ziel
- Coverage allein reicht nicht aus
  - Tests müssen den Rückgabewert verifizieren

QUELLEN

- Bild: TDD Lifecycle; CC BY-SA 4.0

  https://en.wikipedia.org/wiki/Test-driven_development
- Bild: test pyramid; Martin Fowler

  https://www.martinfowler.com/bliki/TestPyramid.html

  https://www.martinfowler.com/bliki/images/testPyramid/test-pyramid.png