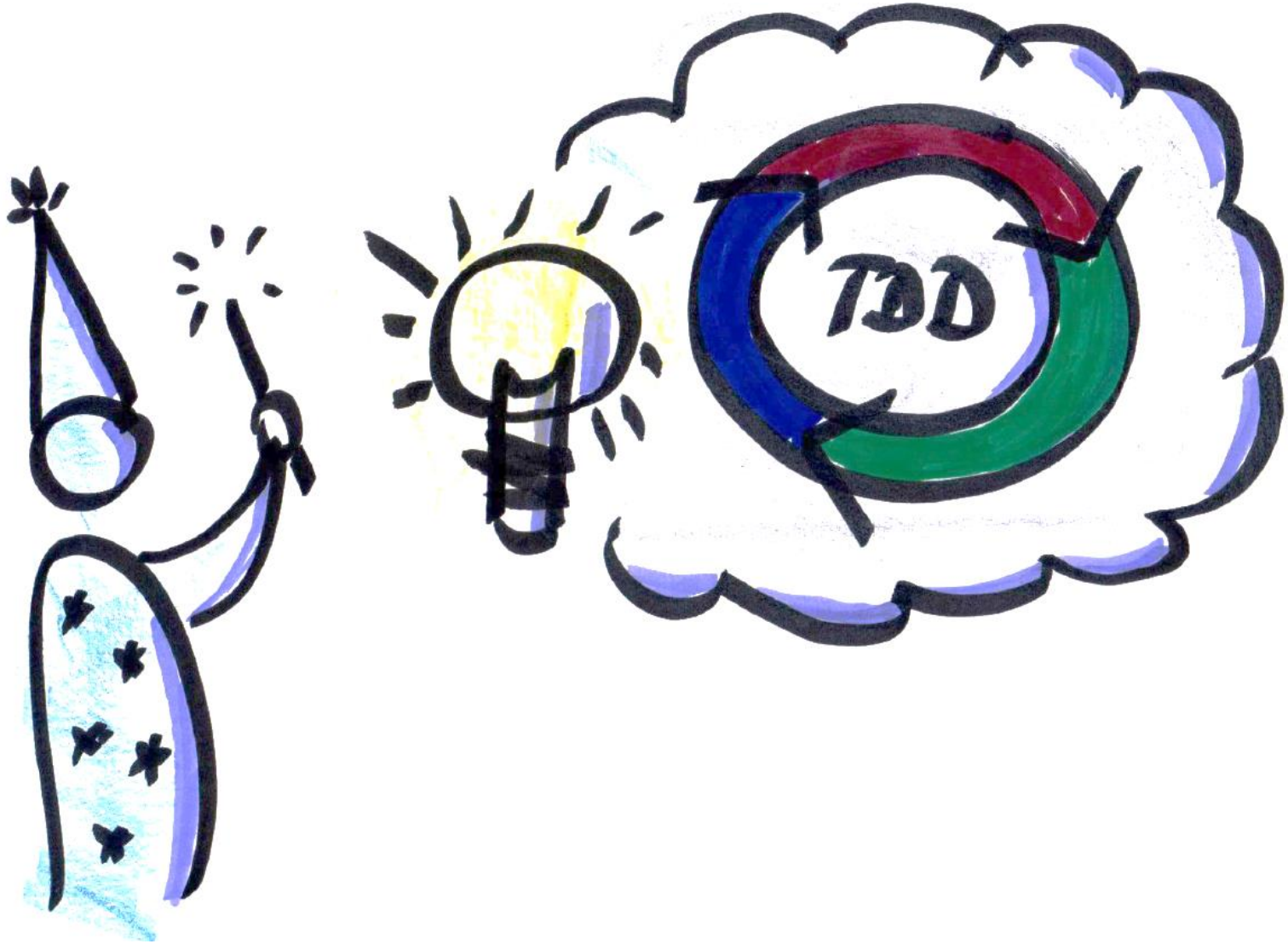
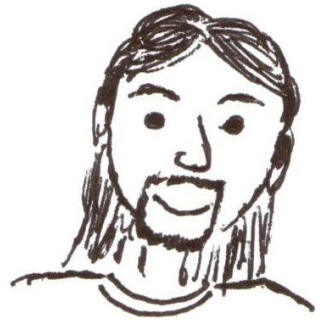
TDD demystified

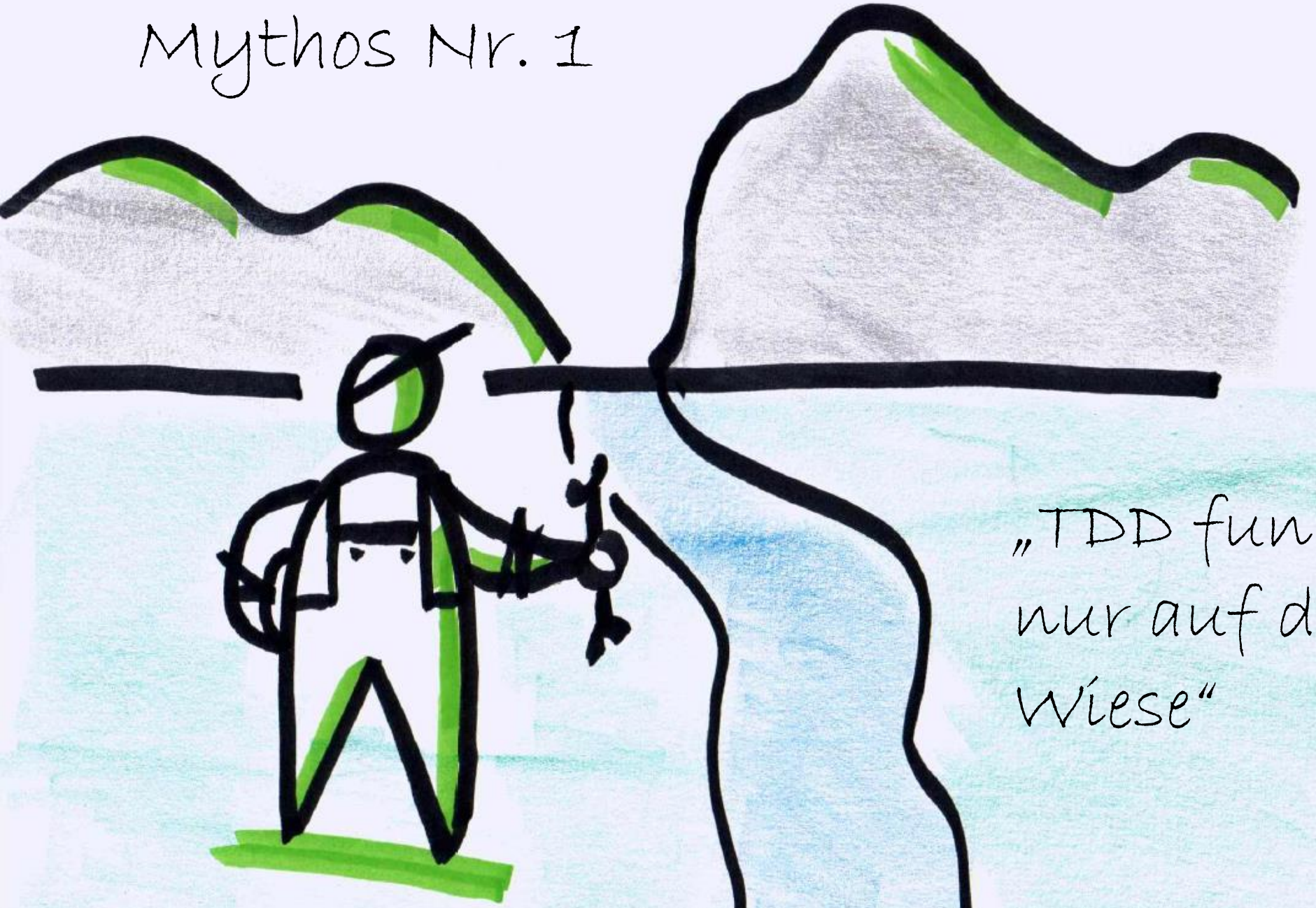




Peter Fichtner
Fiducia & GAD IT AG
@petfic

Tilmann Glaser
It-economics GmbH
@TGnowledgy

Mythos Nr. 1



„TDD funktioniert nur auf der grünen Wiese“

Mythos Nr. 2

„TDD geht nur wenn man schon genau weiß, was man braucht und wie man es umsetzt“

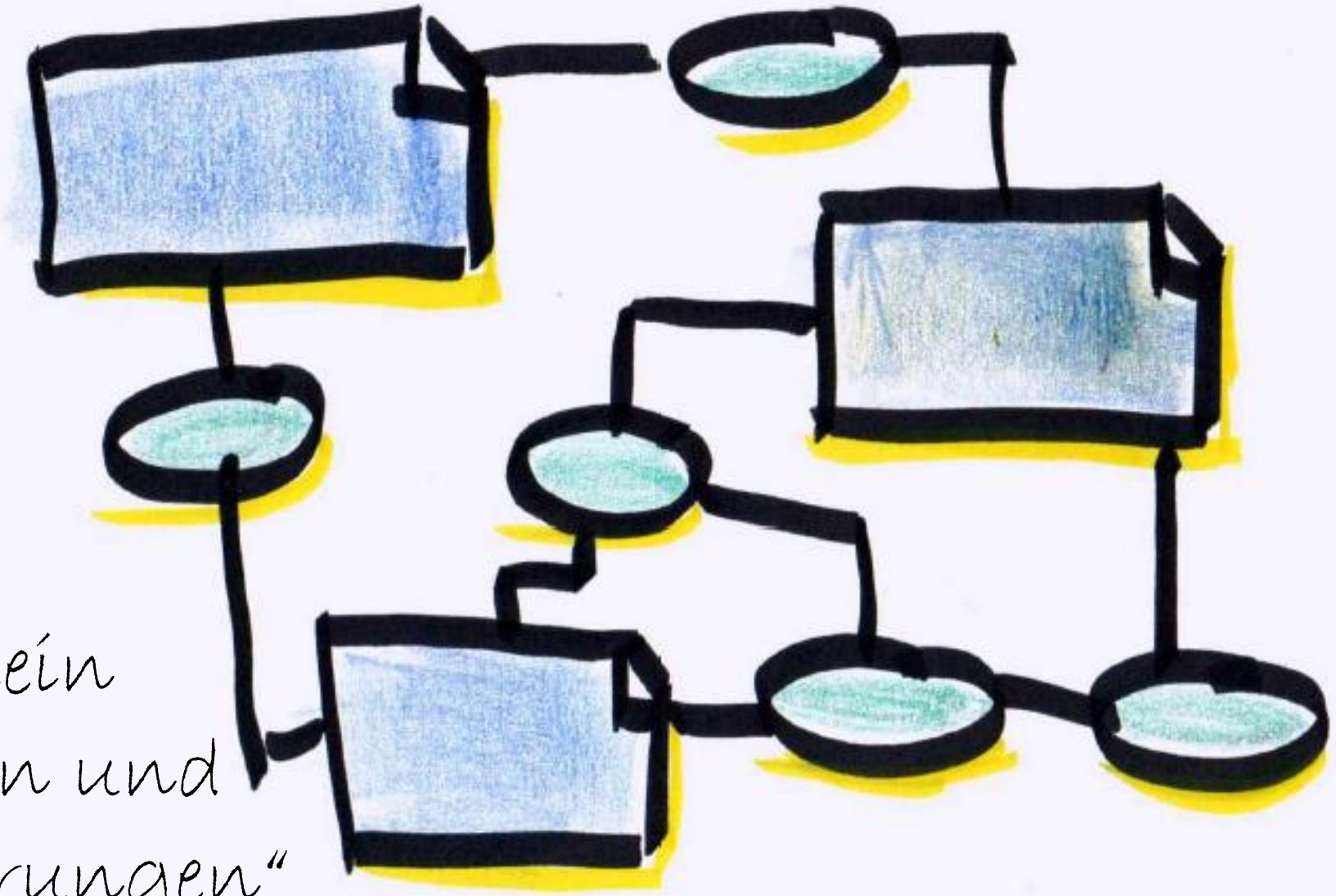
Mythos Nr. 3

„Mit TDD bin ich zu langsam.
Wir haben aber Lieferdruck“

Mythos Nr. 4

„Der Aufwand für die vielen Tests ist viel zu hoch für den wenigen Nutzen."

Mythos Nr. 5



„TDD zementiert mein Anwendungsdesign und verhindert Optimierungen“

Schlussfolgerung

„TDD ist ganz nett für Coding Dojos.

Aber für echte Entwicklung nicht praktikabel“

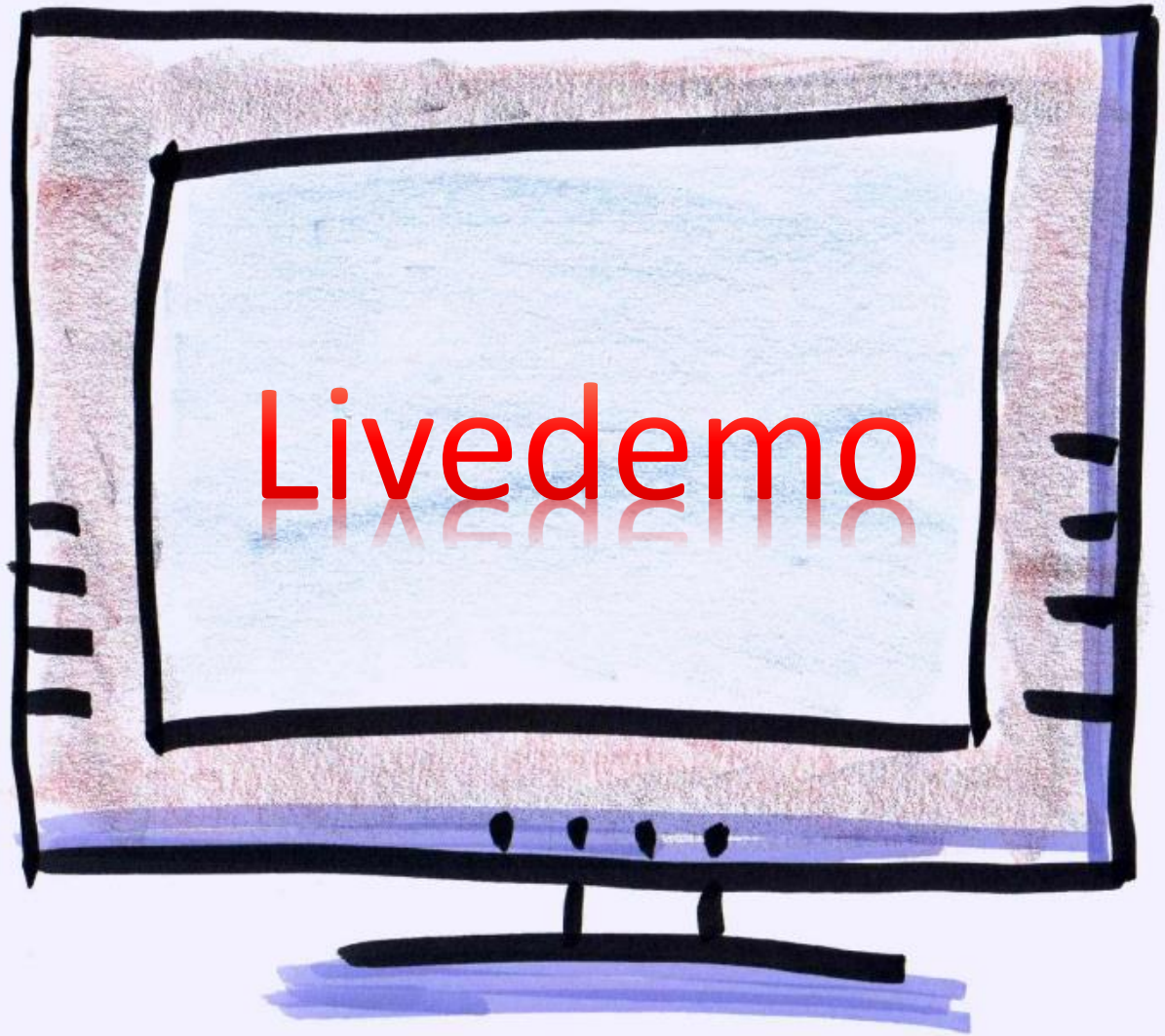
Woher kommen diese Mythen

- Coding Dojos vermitteln falsches Verständnis von TDD
- „Unit“ zu feingranular gewählt
- Big Design Upfront
- Strukturhörigkeit

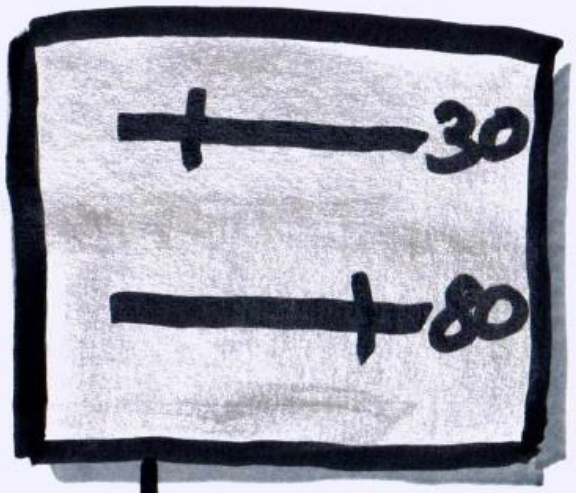
Ja, aber...

... wie kann man TDD angehen, damit es praktikabel wird?

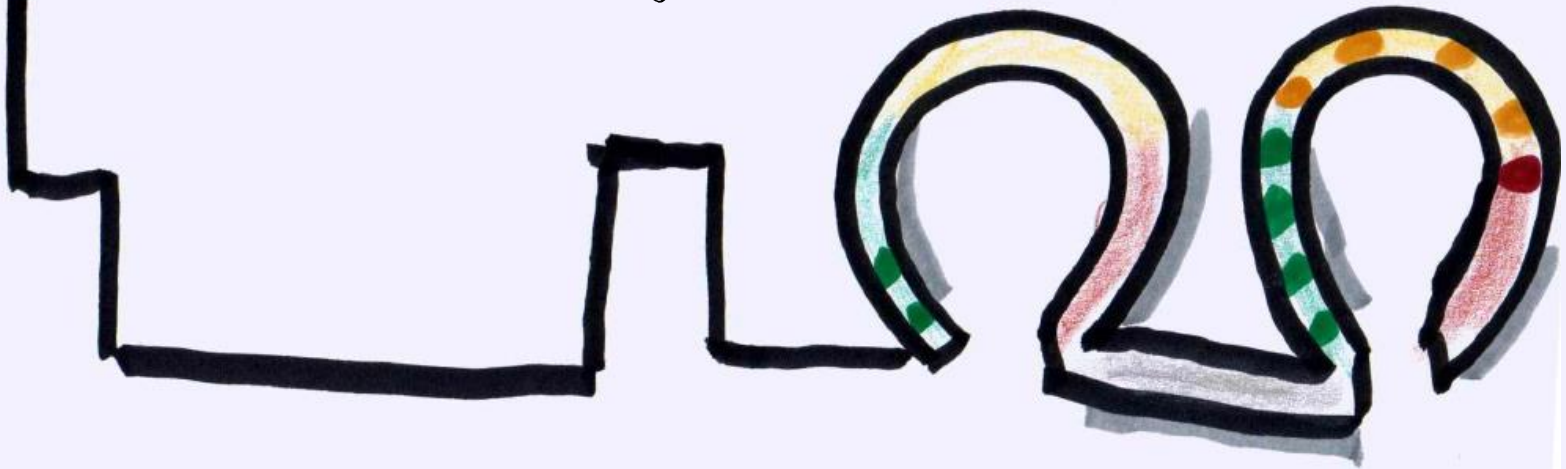
„The Real“ Test Driven Development



„The Real“ Test Driven Development



- Pro Schieberegler soll der jeweils aktuelle Wert mit je einem LED Ring dargestellt werden
  - Die LEDs der zwei Ringe sind logisch in Reihe geschaltet (Ring 1: LED 1-16, Ring 2: LED 17-32)
  - Für jede einzelne LED muss der anzuzeigende Farbwert an die Hardware gesendet werden
- Zu je einem Drittel sind die Farben je Ring grün, gelb und rot
- Die Richtung der Anzeige (im oder gegen den Uhrzeigersinn) soll je Ring konfigurierbar sein



Entwicklungsstufe 1

- „TDD as if you meant it“
  - Basisfunktionalität in Tests formuliert.
  - 9 Tests, 10 Zeilen Produktivcode
  - Kein Klassendesign
  - Produktivcode ist noch in der Testklasse

- GitHub-Link
  - https://github.com/hellman-and-hero/TDD-demystified/tree/stand1/

Entwicklungsstufe 2

- Erstes Refactoring
  - Aus 4 fix vorgegebenen LED boolean-Flags wird ein boolean-Array
  - 9 Tests, 5 Zeilen Produktivcode
  - Kein Klassendesign
  - Produktivcode ist noch in der Testklasse

- GitHub-Link
  - https://github.com/hellman-and-hero/TDD-demystified/tree/stand2

Entwicklungsstufe 3

- Zweites Refactoring
  - Einführung eines Klassendesign
    - RgbLedRing als Service um LEDs zu schalten
    - RgbLedRing-Service nutzt ein MqttClient-Interface um einen Test-Client per Dependency Injection mitzugeben
  - Anpassung der Tests an das Klassendesign
    - Nutz minimale Aufwände durch vorherige Kapselung nach given/when/then
  - 9 Tests, 42 Zeilen Produktivcode
  - Starke Abhängigkeit zu Klassen aus „org.eclipse.paho.client.mqttv3“

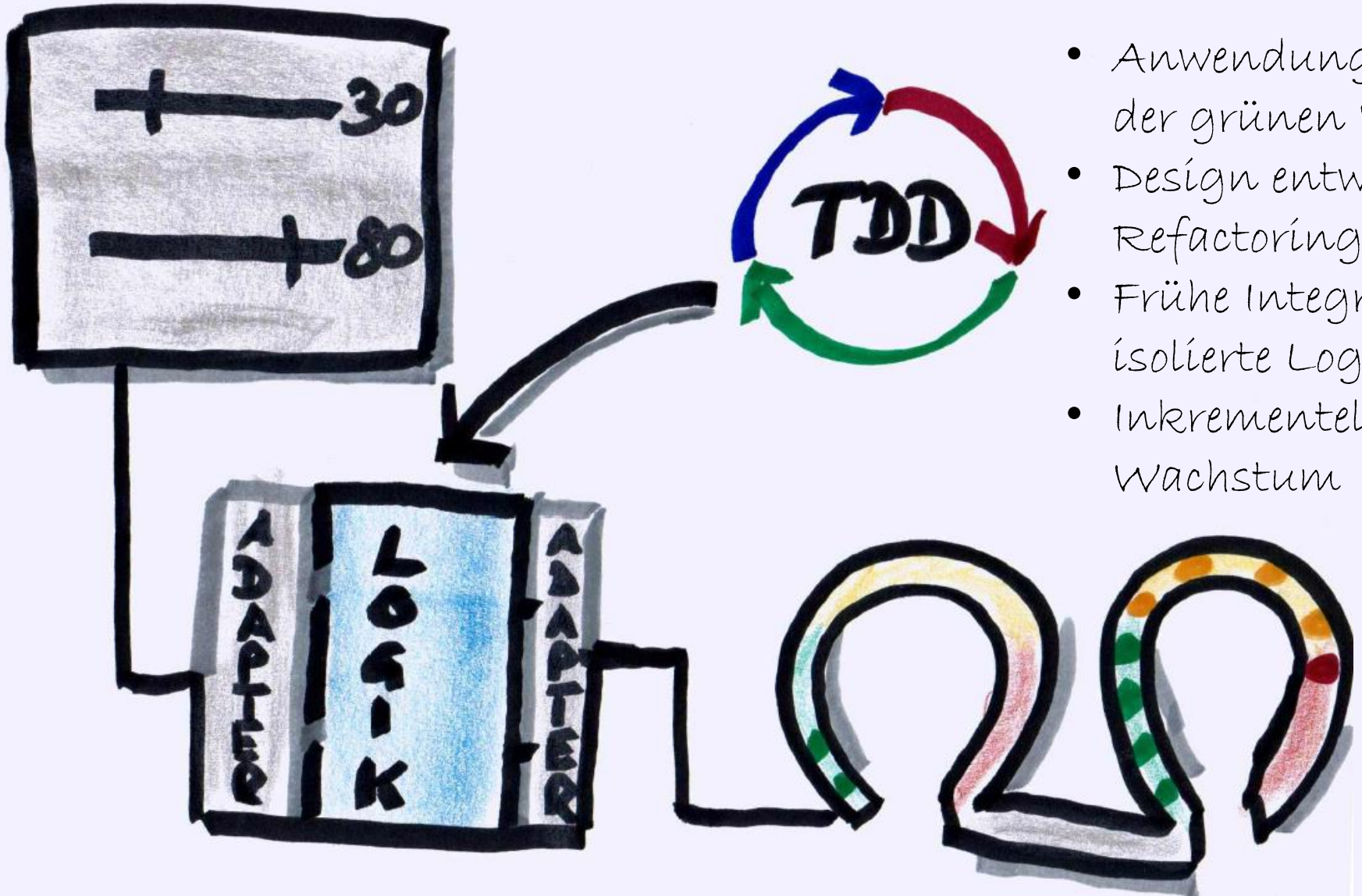
- GitHub-Link
  - https://github.com/hellman-and-hero/TDD-demystified/tree/stand3/

Entwicklungsstufe 4

- Finaler Schliff
  - Kapselung des MqttClients hinter allgemeinem Adapter
  - Implementierung eines echten MqttClient
    - TDD über Integrationstest
  - Erweiterung der Funktionalität um unterschiedliche Farben
  - 12 Tests, 77 Zeilen Produktivcode
  - Ende-zu-Ende Nutzbarer „LevelToLedMeter“-Service
  - Kann nun direkt in das bestehende Umfeld integriert werden

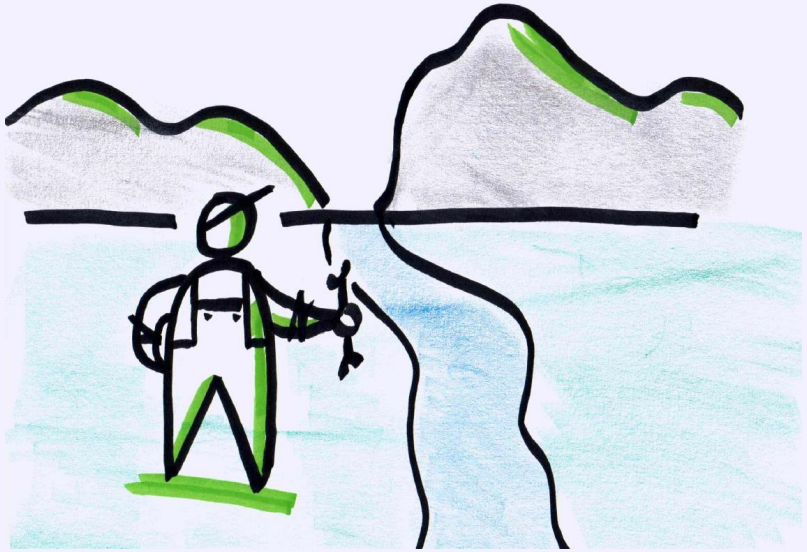
- GitHub-Link
  - https://github.com/hellman-and-hero/TDD-demystified/tree/master

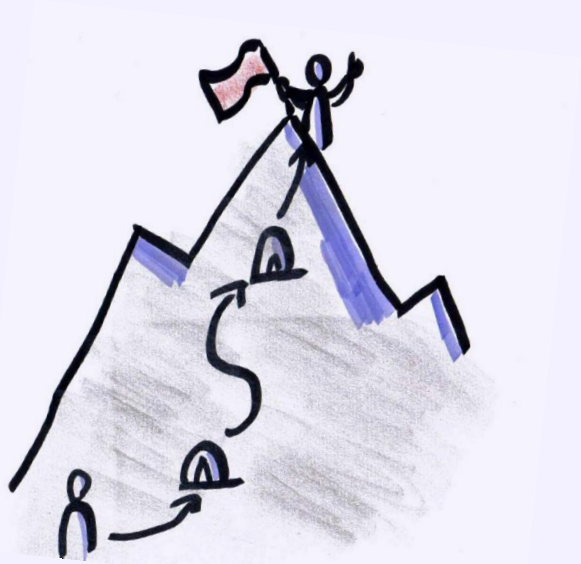
„The Real“ Test Driven Development

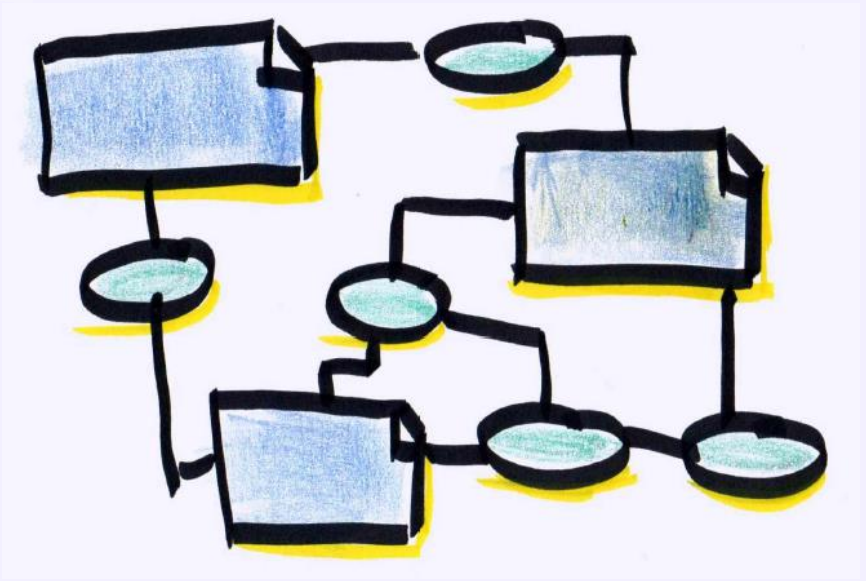


- Anwendung wird scheinbar auf der grünen Wiese entwickelt
- Design entwickelt sich aus dem Refactoring
- Frühe Integration und trotzdem isolierte Logik
- Inkrementelles Feature-Wachstum

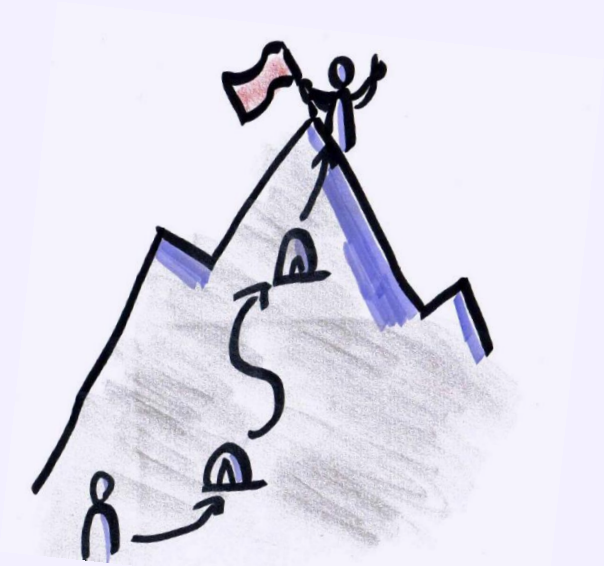
Demystified?

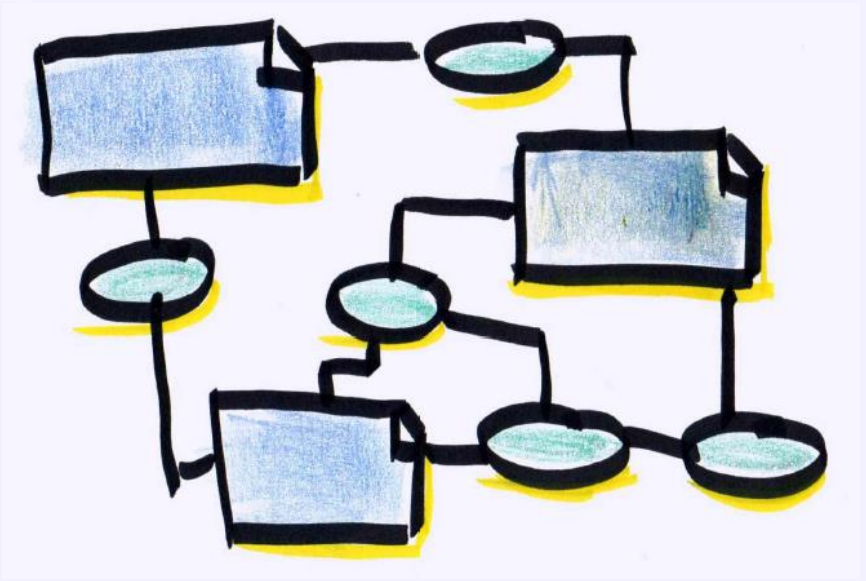




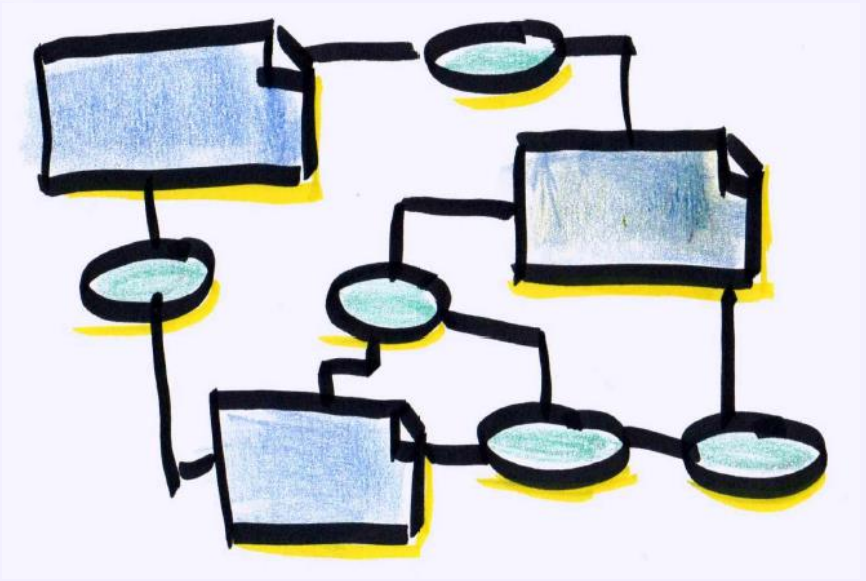


Demystified?

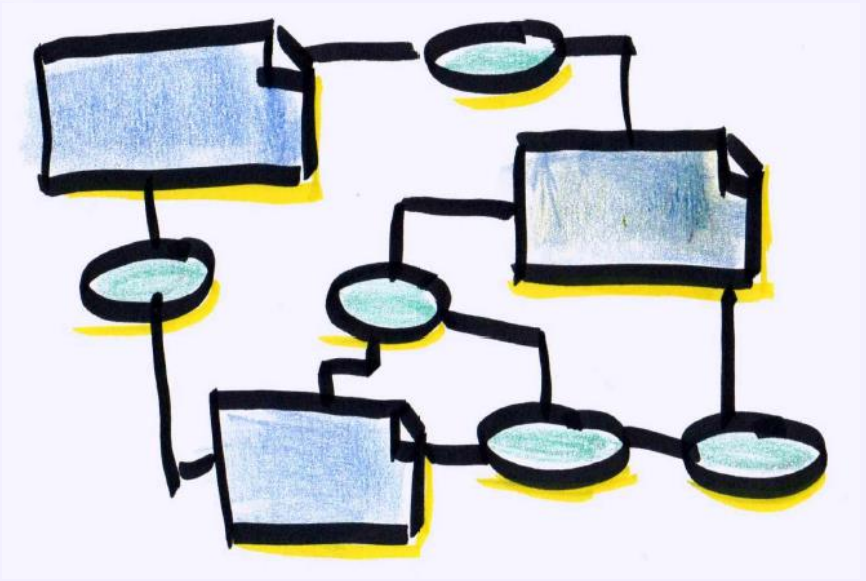




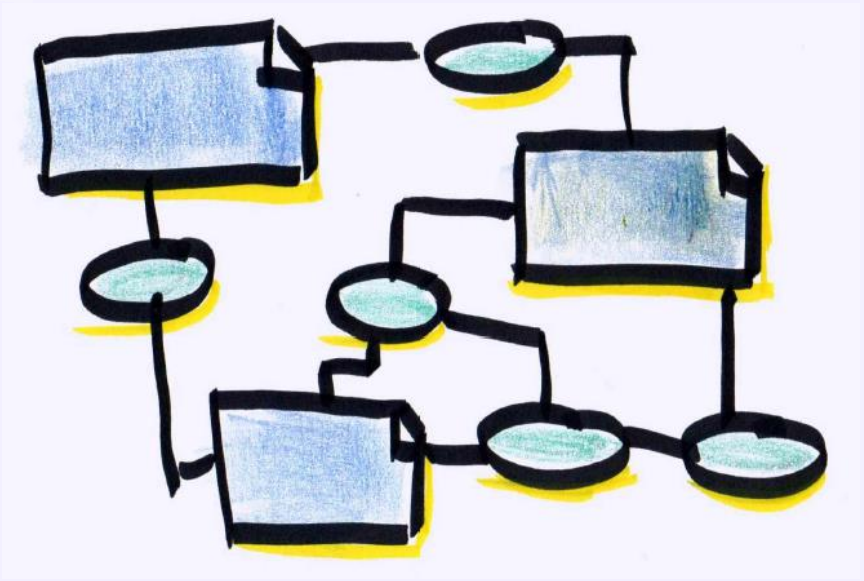
Demystified?



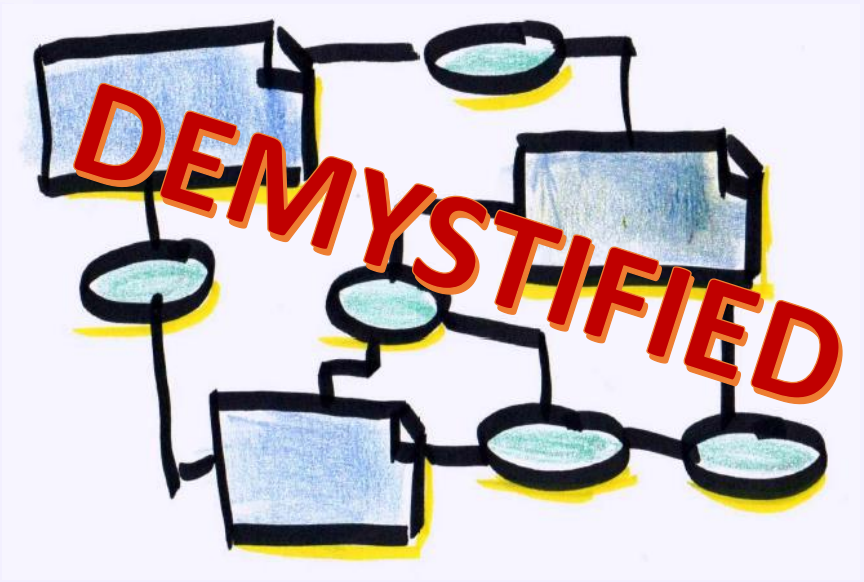
Demystified?



Demystified?



Demystified?



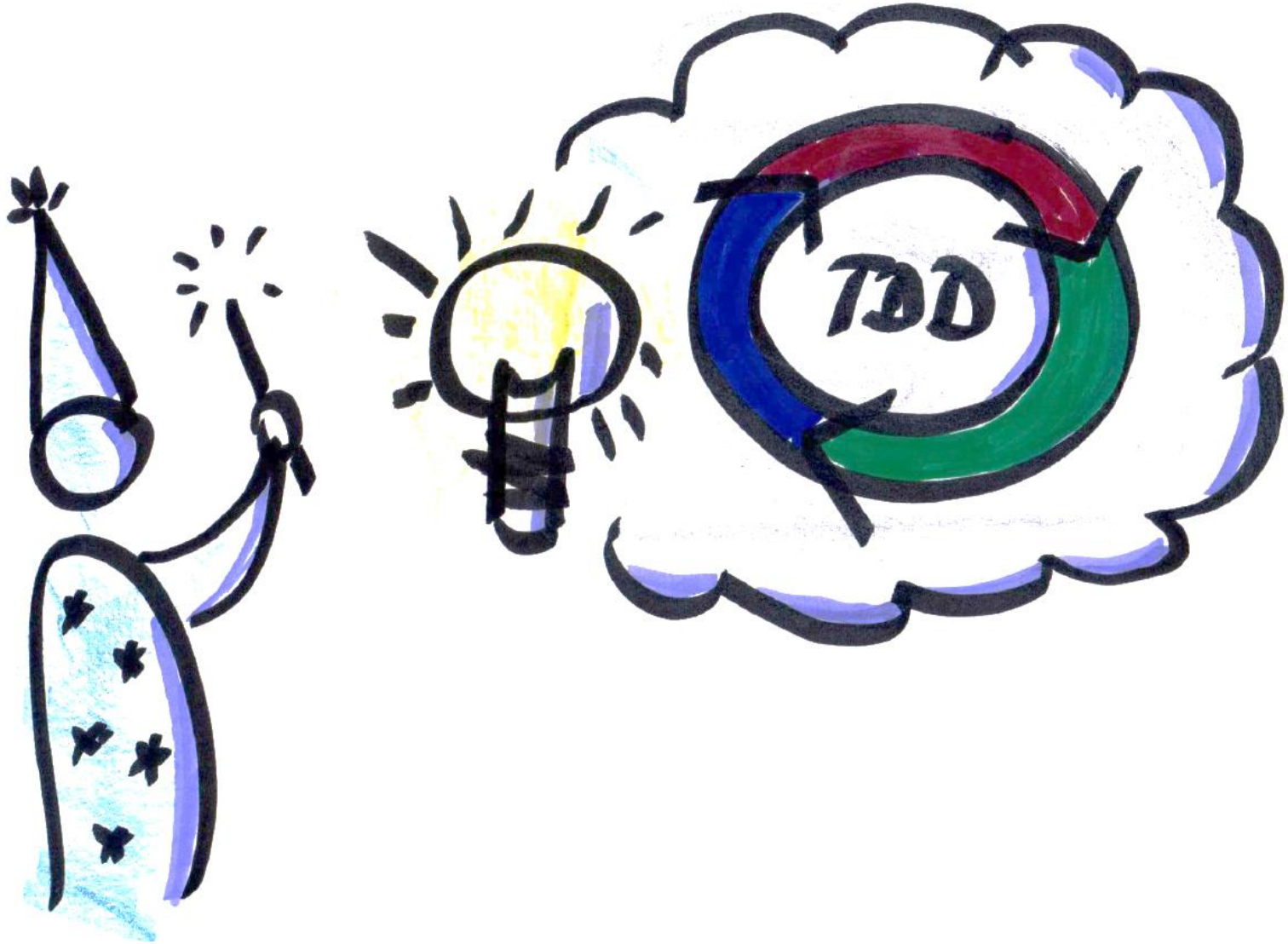
Fazit

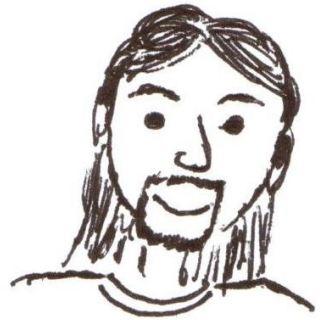
TDD ist ein irreführender Begriff

Gutes Anwendungsdesign, mit nachhaltiger Architektur ist auch ohne TDD möglich

Aber mit richtigem TDD ist es schwer, gegen die Regeln des guten Designs zu verstoßen

TDD demystified





Peter Fichtner
Fiducia & GAD IT AG
@petfic

Tilmann Glaser
It-economics GmbH
@TGnowledgy