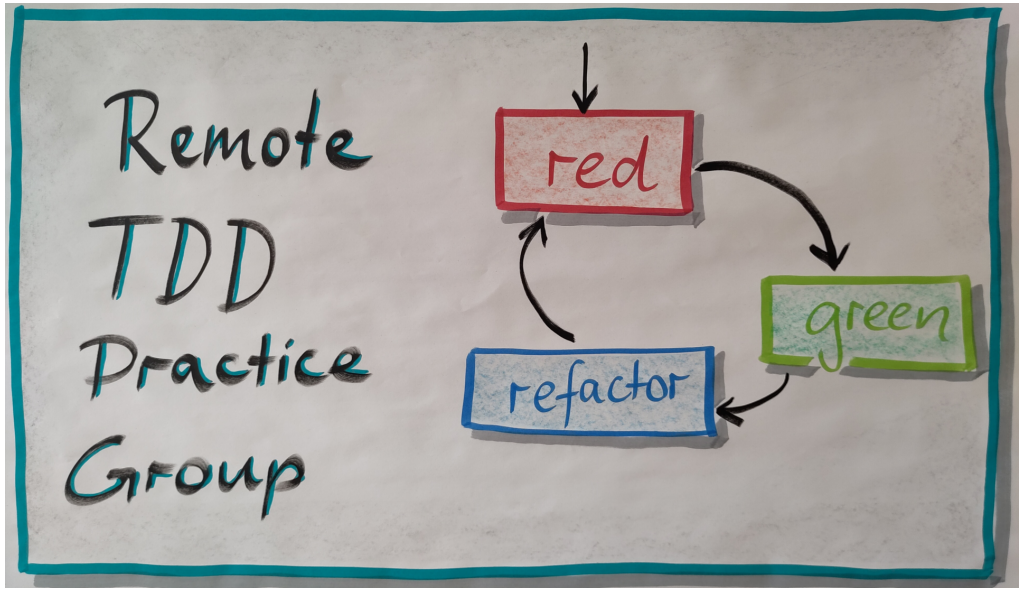
technical agile coach



Handling Dependencies with Test Doubles

14.5.2024

Armin Troy - Technical Agile Coach - www.technicalagile.coach

Dependencies

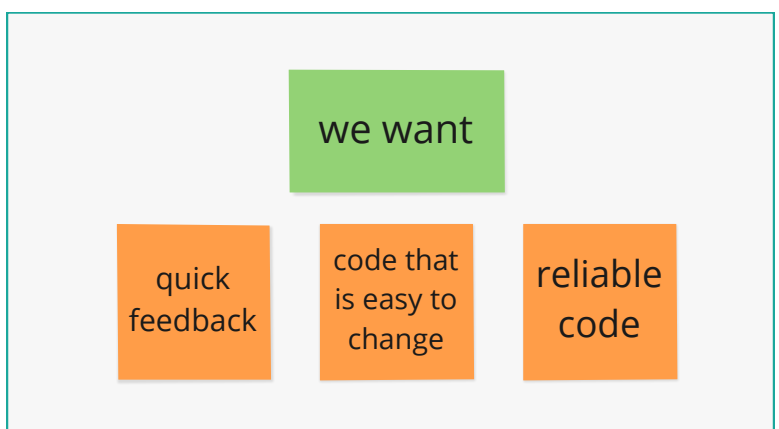
technical agile coach

dependencies on

- classes or functions
- libraries
- micro-services
- external systems, e.g. database, web server, mail server, ...

why are dependencies bad for TDD?

- can slow down tests
- more difficult to write tests and maintain them
- refactoring can get harder
  - when dependencies change also the tests have to be changed
- harder to test components in isolation, e.g. difficult to test error handling
- tests get fragile
  - when underlying dependencies change
  - sensitive to behaviour of external systems



Dependency Injection

technical agile coach

hard-coded dependency

```python
class Database:
    def __init__(self, host, port):
        self.host = host
        self.port = port

    def connect(self):
        print(f"Connecting to database at {self.host}:{self.port}")

class UserService:
    def __init__(self):
        self.db = Database(host="localhost", port=5432) # <- dependency!

    def get_user(self, user_id):
        self.db.connect()
        return f"Getting user with id {user_id}"
```

dependency is "injected" as argument

```python
class UserServiceWithDI:
    def __init__(self, db):
        self.db = db

    def get_user(self, user_id):
        self.db.connect()
        return f"Getting user with id {user_id}"
```

fake instead of real database

```python
class DatabaseFake:
    def __init__(self):
        pass

    def connect(self):
        print(f"Connecting to fake database")
```

test cases with/without dependency injection

```python
def test_db():
    user_service = UserService() # <- has fixed internal dependency on database
    assert user_service.get_user(user_id=123) == "Getting user with id 123"

def test_db_with_dependency_injection():
    db = Database(host="localhost", port=5432)
    user_service = UserServiceWithDI(db) # <- uses the same database but dependency is "injected"
    assert user_service.get_user(user_id=123) == "Getting user with id 123"

def test_fake_db():
    fake_db = DatabaseFake() # <- fake instead of the real database ()
    user_service = UserServiceWithDI(fake_db)
    assert user_service.get_user(user_id=123) == "Getting user with id 123"
```

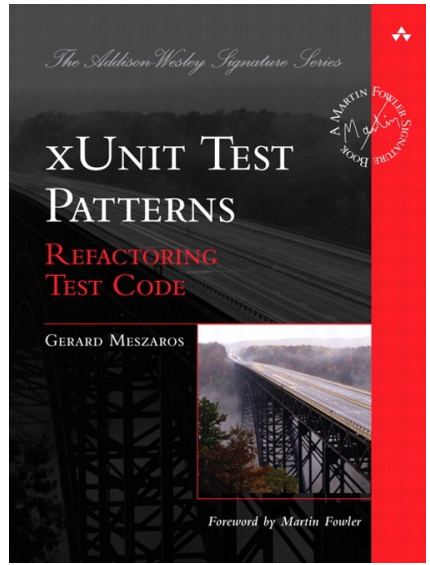
Test Doubles

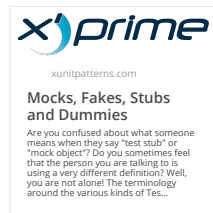
generic term for any kind of pretend objects used in place of a real object for testing purposes

technical agile coach

Dummy Objects	are passed around but never actually used. Usually they are just used to fill parameter lists.
Fakes	actually have working implementations, but usually take some shortcut which makes them unsuitable for production (an in memory database is a good example).
Stubs	provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
Spies	are stubs that also record some information based on how they were called. One form of this might be an email service that records how many messages it has sent.
Mocks	objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

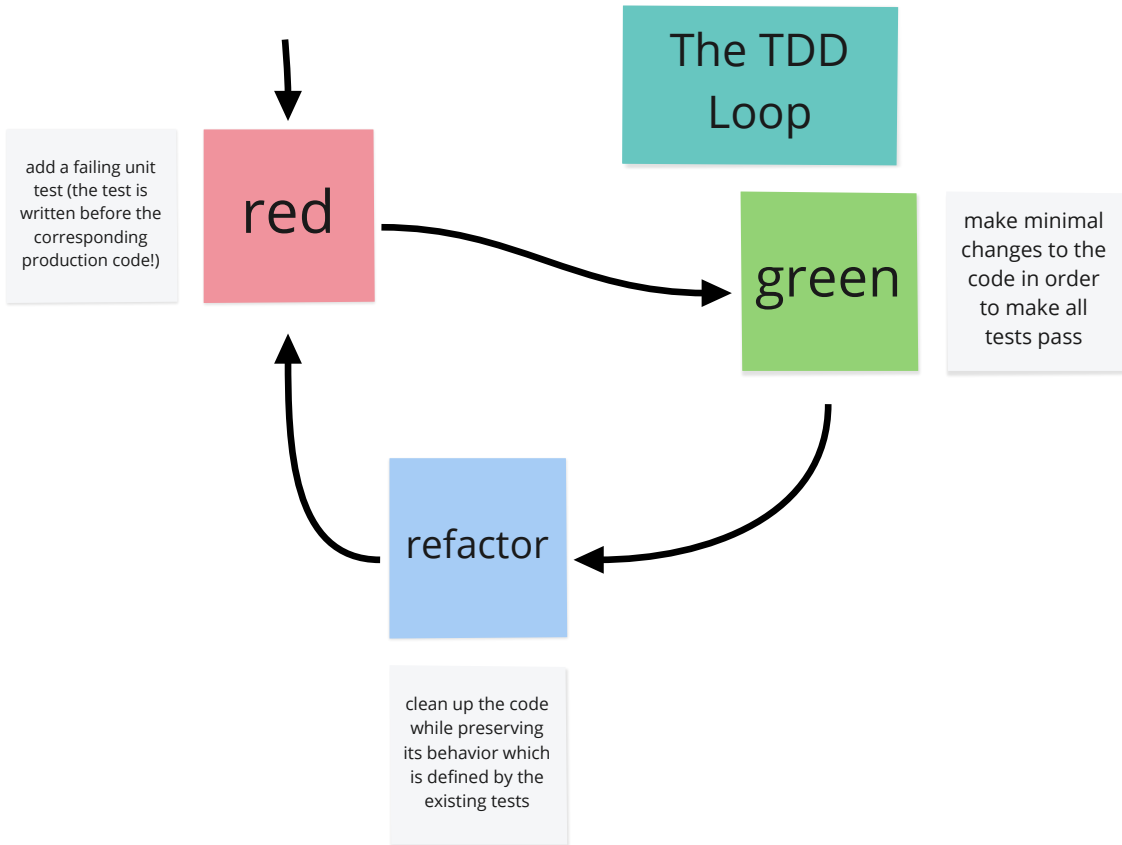
Definitions from Gerard Meszaros' book:





Fundamentals of Test-Driven Development

technical agile coach



Pair Programming and Ensemble Programming

technical agile coach

Pair Programming

2 persons are programming together on one computer

Ensemble Programming (Mob Programming)

>2 persons are programming together on one computer

Roles

Driver

does all the typing

Navigator

gives directions to the driver / decides what to code next

different patterns/rules when to switch roles, ...