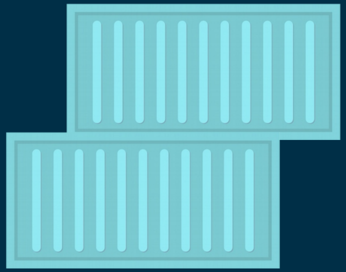
Testing

1

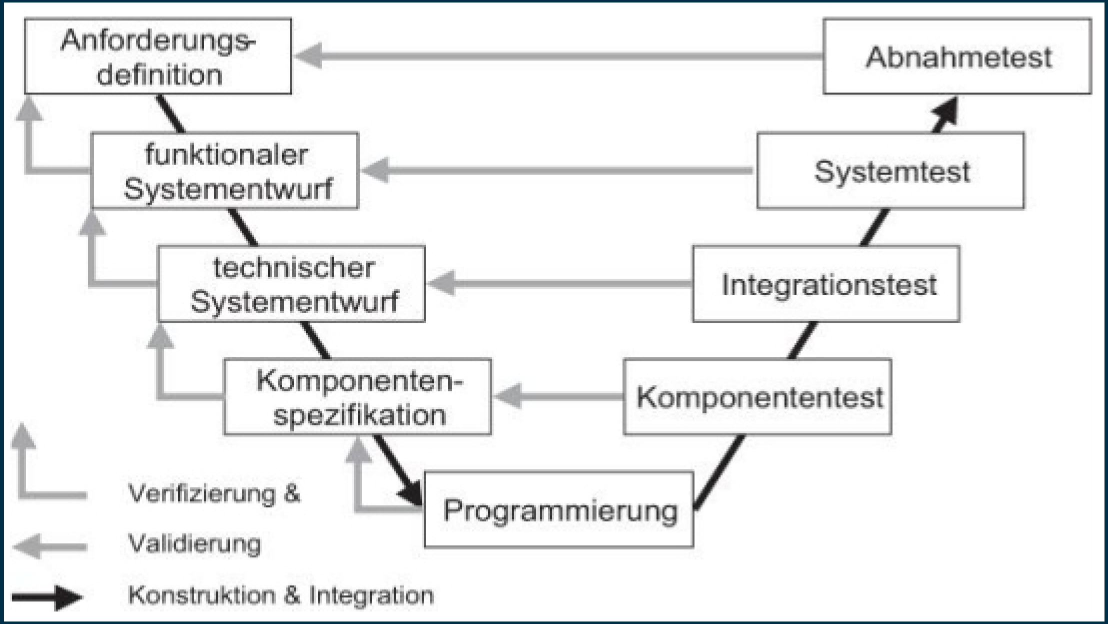


Testing basics

Why testing?

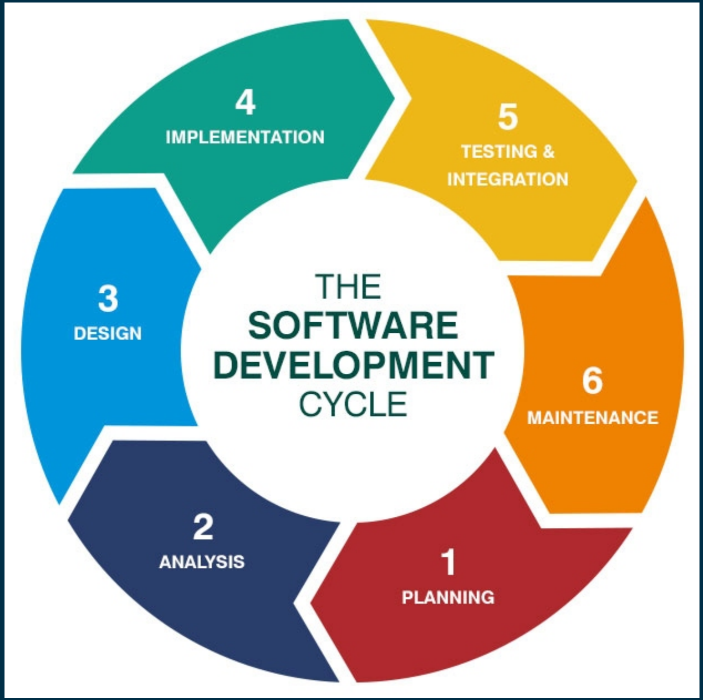
- prove that your code is working
- automated and fast feedback
- confidence for developers
- base for continuous deployment

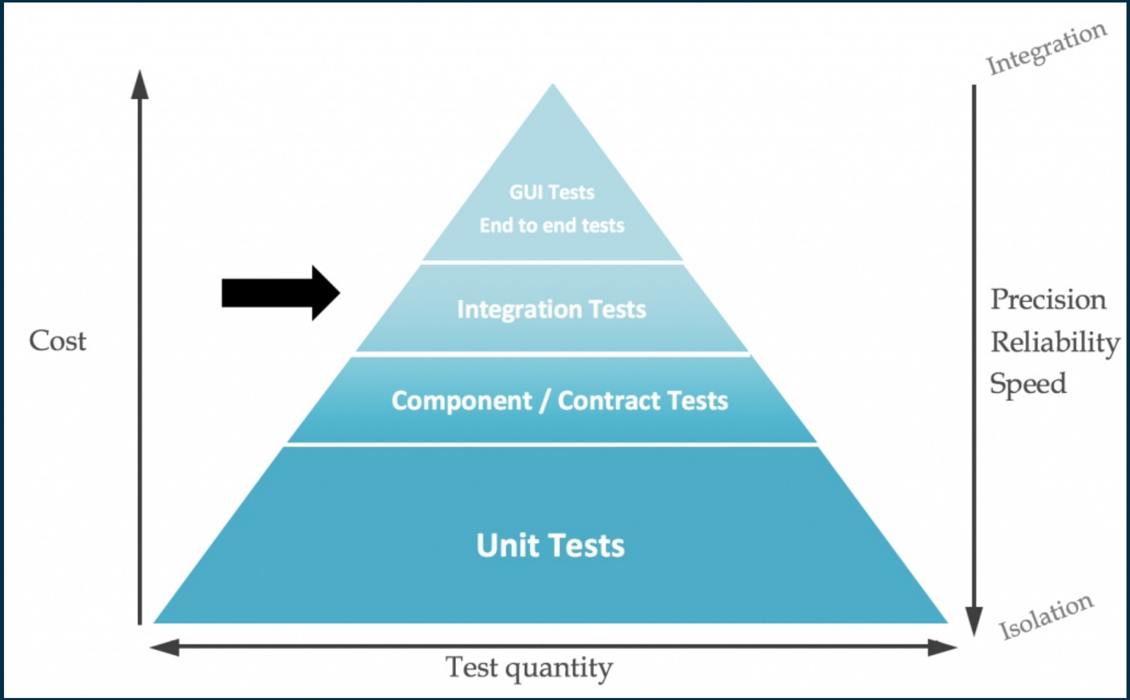
Testing levels V-model



Source: ISTQB / iSAQB

Testing levels





Source: Le Blog des Octos

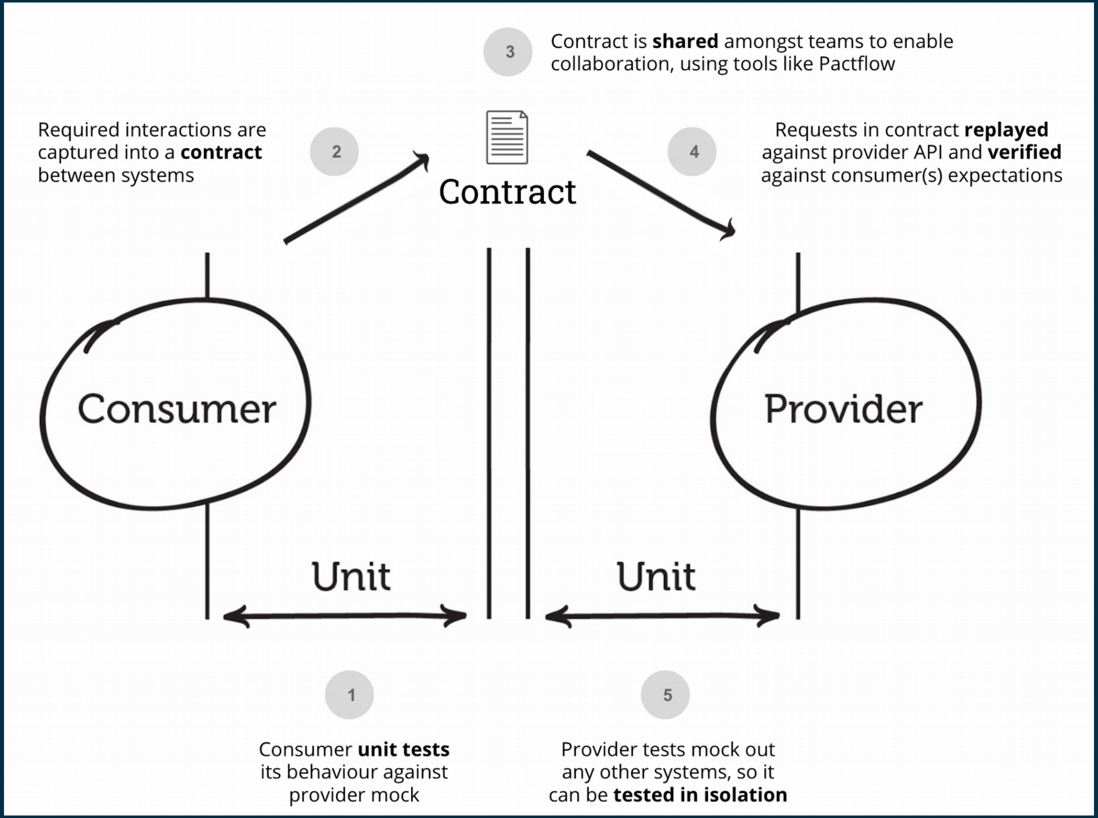
Unit tests

- biggest amount of tests
- code coverage
- responsibility of the developer
- run locally
- no dependencies

Component tests

- testing of the component
- mocked dependencies
- responsibility of the development team
- Consumer Driven Contract Testing

Contract Testing



3
Contract is **shared** amongst teams to enable collaboration, using tools like Pactflow
Required interactions are captured into a **contract** between systems
2
Contract
4
Requests in contract **replayed** against provider API and **verified** against consumer(s) expectations
Consumer
Provider
Unit
Unit
1
5
Consumer **unit tests** its behaviour against provider mock
Provider tests mock out any other systems, so it can be **tested in isolation**

Source: Pact

Integration tests

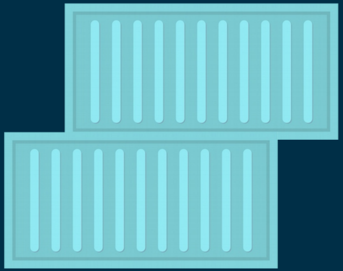
- run on tested components
- testing of the components working together
- real dependencies
- evaluate the compliance of a system or component with specified functional requirements
- goal: find problems in component communication or collaboration

System tests (End to End)

- feature testing of the whole system
- acceptance tests (UAT)
  - automated or manual
- goal: show that the whole system acts as defined in the requirements

2

Advanced testing



Quality tests

- performance tests
- regression tests
- ISO/IEC 9126
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

Quality and security analysis

- Static Code Analysis (SCA)
- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Dependency Checks
- Licenses Scan
- Security scan of the deployable artifact

3

Test Driven Development (TDD)

Tests first

- write tests before start coding
- more than one tests can fail at the same time
- tests must not be written by the developer that implements the new feature

TDD following Kent Beck

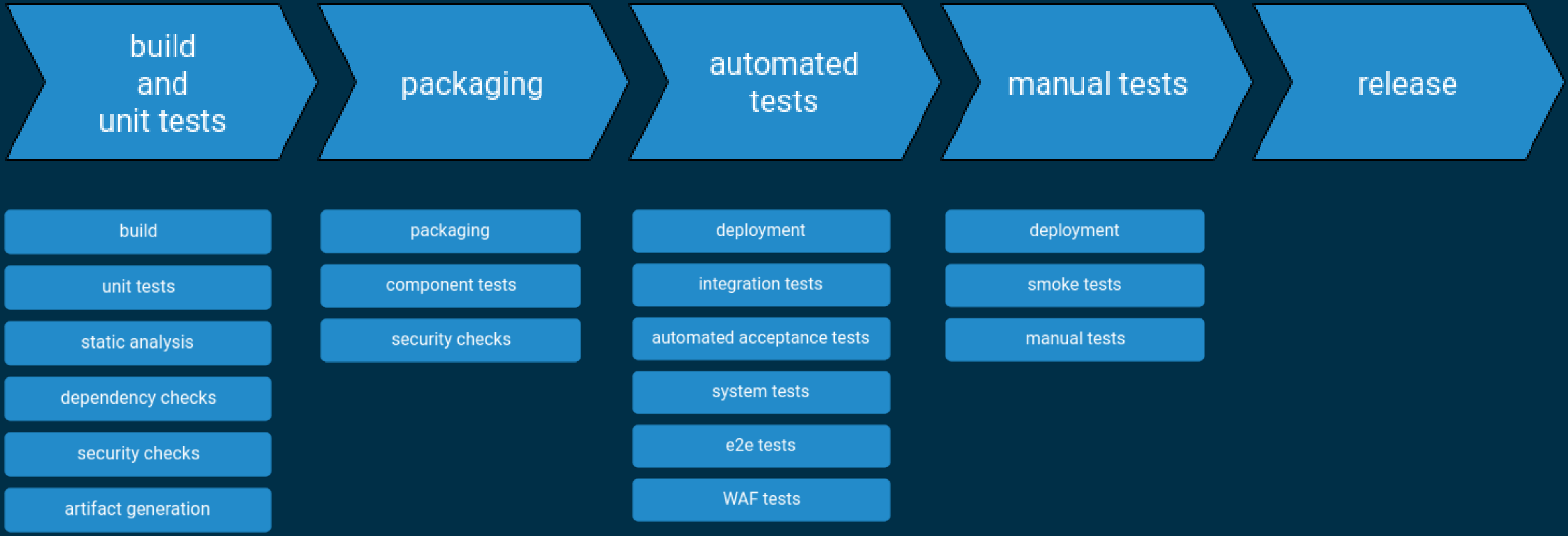
red-green-refactor iterations

- red: write unit tests for new feature (tests fail)
- green: implement feature with minimal work (tests pass)
- refactor: clean code, remove duplicates, follow coding guidelines

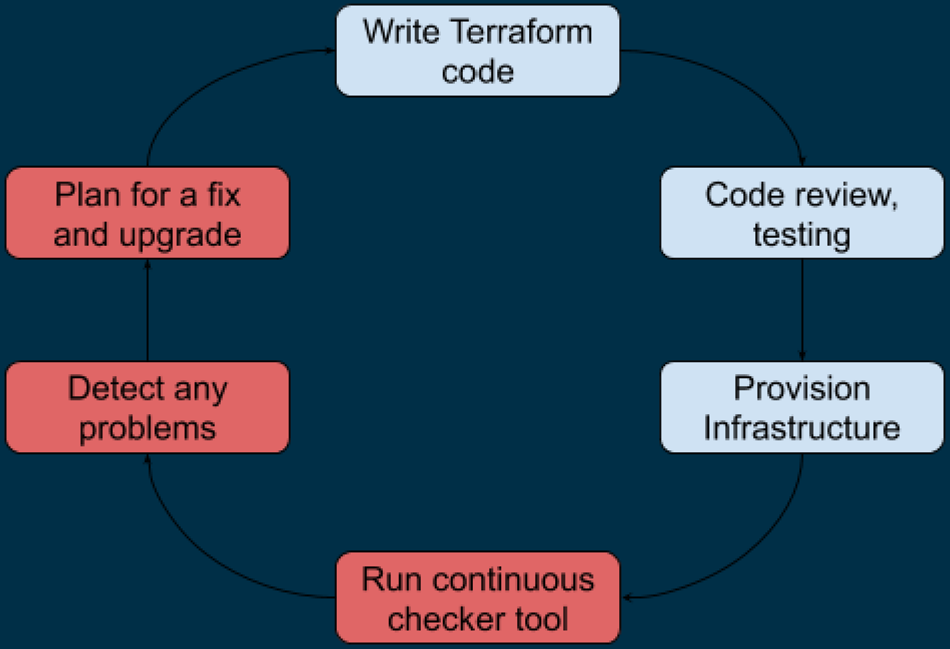
4

Testing inside the delivery pipeline

Delivery Pipeline



Infrastructure



Source: DuploCloud