Testovací pyramida

Testovací pyramida je koncept v softwarovém inženýrství, který vizualizuje **doporučené poměry a hierarchii různých typů testů** v softwarovém projektu. Tento model poprvé formuloval **Mike Cohn** ve své knize *Succeeding with Agile* (2009) a od té doby se stal základním principem pro budování udržitelné a efektivní testovací strategie.

Cílem testovací pyramidy je:

- minimalizovat náklady na testování,
- maximalizovat rychlost zpětné vazby,
- zvýšit spolehlivost testovací sady,
- a podpořit rychlý vývoj s vysokou kvalitou.

Struktura testovací pyramidy

Pyramida se skládá ze **tří hlavních vrstev**, uspořádaných od nejrychlejších a nejlevnějších testů (spodek) po nejpomalejší a nejnákladnější (vrchol):

1. Jednotkové testy (Unit Tests) – základna pyramidy

- **Co testují**: Izolované jednotky kódu (např. funkce, metody, třídy).
- **Rychlost**: Milisekundy – spouští se běžně stovky až tisíce za sekundu.
- **Náklady na údržbu**: Nízké.
- **Stabilita**: Vysoká – selhávají jen při změně logiky.
- **Doporučený podíl**: **70 % a více** všech automatizovaných testů.

> **Dobrý jednotkový test** je rychlý, izolovaný, deterministický a snadno čitelný.

2. Integrační testy (Integration Tests) – střední vrstva

- **Co testují**: Interakce mezi více komponentami nebo systémy (např. volání databáze, API, služby třetích stran).
- **Rychlost**: Desítky milisekund až sekundy.
- **Náklady na údržbu**: Střední – závisí na složitosti integrací.
- **Stabilita**: Střední – mohou selhat kvůli externím závislostem.
- **Doporučený podíl**: **20–25 %** testů.

> Integrační testy ověřují, že části systému „spolu hrají“ správně.

3. End-to-end (E2E) testy – vrchol pyramidy

- **Co testují**: Kompletní uživatelské scénáře napříč celou aplikací (např. „uživatel se přihlásí, přidá zboží do košíku a zaplatí“).
- **Rychlost**: Sekundy až minuty.
- **Náklady na údržbu**: Vysoké – citlivé na změny UI.

- **Stabilita**: Nízká – často selhávají kvůli dočasným problémům (časování, síť).
- **Doporučený podíl**: **5–10 %** testů.

> E2E testy simulují skutečného uživatele, ale **nejsou náhradou za jednotkové testy**!

Vizuální znázornění pyramidy

```
          ┌─────────────────────┐
          │  End-to-End testy   │ ← málo, pomalé, drahé
          └─────────────────────┘
        ┌─────────────────────────┐
        │   Integrační testy      │ ← střední množství
        └─────────────────────────┘
┌─────────────────────────────────────┐
│        Jednotkové testy             │ ← hodně, rychlé, levné
└─────────────────────────────────────┘
```

Konkrétní příklady testů

Jednotkový test v Pythonu (s pytest)

Soubor: `calculator.py`

```python
def divide(a, b):
    if b == 0:
        raise ValueError("Dělení nulou není povoleno")
    return a / b
```

Soubor: `test_calculator.py`

```python
import pytest
from calculator import divide

def test_divide_positive_numbers():
    assert divide(10, 2) == 5.0

def test_divide_by_zero_raises_error():
    with pytest.raises(ValueError, match="Dělení nulou není povoleno"):
        divide(5, 0)
```

> Rychlé, izolované, bez závislostí – ideální jednotkový test.

Jednotkový test v Javě (s JUnit 5)

Soubor: `BankAccount.java`

```
public class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void withdraw(double amount) {
        if (amount > balance) {
            throw new IllegalArgumentException("Nedostatek prostředků");
        }
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

Soubor: `BankAccountTest.java`

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class BankAccountTest {

    @Test
    void withdraw_ValidAmount_ReducesBalance() {
        BankAccount account = new BankAccount(100.0);
        account.withdraw(30.0);
        assertEquals(70.0, account.getBalance());
    }

    @Test
    void withdraw_InsufficientFunds_ThrowsException() {
        BankAccount account = new BankAccount(50.0);
        assertThrows(IllegalArgumentException.class, () ->
account.withdraw(60.0));
    }
}
```

Integrační test v Pythonu (s databází pomocí Testcontainers)

```
import pytest
from sqlalchemy import create_engine, text
from myapp.models import User

@pytest.fixture(scope="module")
def db_engine():
```

```
    # Spustí Docker kontejner s PostgreSQL (přes testcontainers)
    with PostgresContainer("postgres:15") as postgres:
        engine = create_engine(postgres.get_connection_url())
        yield engine

def test_user_persistence(db_engine):
    # Vložíme uživatele do DB
    with db_engine.connect() as conn:
        conn.execute(text("INSERT INTO users (name) VALUES ('Alice')"))
        result = conn.execute(text("SELECT name FROM users WHERE name =
'Alice'"))
        assert result.fetchone()[0] == "Alice"
```

E2E test v Javě (s Playwright)

```
import com.microsoft.playwright.*;
import org.junit.jupiter.api.*;

public class LoginE2ETest {
    static Playwright playwright;
    static Browser browser;

    @BeforeAll
    static void launchBrowser() {
        playwright = Playwright.create();
        browser = playwright.chromium().launch();
    }

    @Test
    void successfulLogin() {
        Page page = browser.newPage();
        page.navigate("https://mojeaplikace.cz/login");
        page.fill("#username", "admin");
        page.fill("#password", "heslo123");
        page.click("button[type='submit']");
        assertTrue(page.url().contains("/dashboard"));
        assertTrue(page.textContent("h1").contains("Vítejte"));
    }

    @AfterAll
    static void closeBrowser() {
        playwright.close();
    }
}
```

Testovací matice pro mikroslužby

V architektuře **mikroslužeb** se klasická pyramida rozšiřuje o **testovací matici**, která zohledňuje:

- **Vertikální vrstvy** (jednotkové → integrační → E2E),
- **Horizontální dimenzi** (testování uvnitř služby vs. mezi službami).

Úroveň testování	Uvnitř služby	Mezi službami (síť)
Jednotkové testy	Test logiky bez závislostí	–
Integrační testy	Test s databází, cache, externí API	**Kontraktové testy** (Pact), **Consumer-Driven Contracts**
E2E / Špičkové testy	–	Kompletní workflow přes více služeb (např. objednávka → platba → doručení)

> **Klíčový princip**: Nejdříve otestujte logiku **uvnitř služby**, pak ověřte **smlouvy mezi službami**, a až nakonec spusťte nákladné E2E testy.

Tento přístup minimalizuje závislost na „živém“ prostředí a zvyšuje izolaci testů.

Proč dodržovat testovací pyramidu?

- **Rychlá zpětná vazba**: Jednotkové testy selžou během sekund, nikoli minut.
- **Nižší náklady**: Údržba 1000 jednotkových testů je levnější než 10 E2E testů.
- **Lepší laditelnost**: Když selže jednotkový test, víte přesně, kde je chyba.
- **Vyšší pokrytí**: Lze snadno pokrýt okrajové případy (edge cases).
- **Bezpečný refaktoring**: Spolehlivá síť jednotkových testů umožňuje bez obav upravovat kód.

Běžné antipatterny

Testovací kužel (Ice-Cream Cone)

- Příliš mnoho manuálních nebo E2E testů,
- Nedostatek jednotkových testů,
- Důsledek: Pomalé buildy, nestabilní testy, nízká důvěra v automatizaci.

Testovací kobliha (Testing Cupcake)

- Tým má jen E2E testy a žádné jednotkové/integrační,
- Vývoj je pomalý, chyby se hledají hodiny.

Přehnaná automatizace UI

- Každý požadavek se testuje E2E testem,
- Vede k „testovací dlužné pasti“ – testy jsou dražší než kód samotný.

Nástroje podle vrstvy

Vrstva	Příklady nástrojů
——–	——————-

Jednotkové testy	JUnit, pytest, NUnit, Jest, Mocha
Integrační testy	Testcontainers, Pact, Postman, REST Assured
E2E testy	Cypress, Playwright, Selenium, WebdriverIO
BDD testy	Cucumber, SpecFlow, Behave (často nad E2E nebo integrační vrstvou)

Související pojmy

- TDD
- BDD
- CI/CD
- Testování softwaru
- Jednotkový test
- End-to-end testování
- Testovací dluh
- Mikroslužby

Externí odkazy

- Martin Fowler – Test Pyramid: https://martinfowler.com/bliki/TestPyramid.html
- Mike Cohn – Original article: https://mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid
- Google Testing Blog – Just Say No to More End-to-End Tests: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html
- Pact.io – Contract testing for microservices: https://pact.io

Viz také

- Testovací strategie
- Testovací coverage
- QA v agilním týmu
- Živá dokumentace