Testování, Unit testování a Dokumentace zdrojového kódu

Téma není složité, všichni jsme to už prošli. Myslím, že bude stačit, aby si každý normálně přečetl dokument jednou a poté vyzkoušel základní praktický příklad s Unit testy. V nejhorším případě mluvte něco o dokumentaci

## Testování softwaru

Testování softwaru je součástí vývoje softwaru. Jedná o zkoumání kvality a odhalování chyb v programech. Co to ale kvalita je? Kvalita je stupeň shody s požadavky.

Většina intuitivně tuší, že ověřit funkčnost je důležité. Dimenzí kvality je ale podstatně více a na mnohé z nich se zapomíná. Zde jsou:

* **Funkčnost** (Functionality) – správné chování funkcí systému, jak je definováno funkční specifikací
* **Použitelnost** (Usability) – zda vůbec a jak lze dosáhnout požadovaného cíle, zda je systém uživatelsky přívětivý, zda se s ním dobře pracuje
* **Spolehlivost** (Reliability) – zda se chová stejně za všech okolností, zvláště po přetížení, nebo po výpadku či chybě, zda tyto stavy umí detekovat a hlásit
* **Výkon** (Performance) – zda systém není pomalý a zvládne větší množství současně pracujících uživatelů, nebo naopak zda si i při splnění všech požadavků na obsluhu uživatelů nebere příliš systémových zdrojů
* **Podpora** (Supportability) – zda se systém dobře instaluje, nemá problémy s cílovými hardwarovými a softwarovými konfiguracemi a další vlastnosti související s údržbou systému a upgradovatelností
* **Bezpečnost** (Security) – zda jsou systém i jeho data bezpečné a mohou jej používat pouze oprávněné osoby
* **Kompatibilita** (Compatibility) – zda je možné jej používat i s ostatními programy a systémy
* **Přenositelnost** (Portability) – zda je možné jej přenášet například na jiný OS

Další způsoby dělení testování jsou podle:

* **Fáze testování**
  + Unit testy: testují nejmenší testovatelnou jednotku kódu a ověřují, zda se chová správně, nezávisle na ostatních částech systému.
  + Modul Testy: testují celé funkční moduly kódu a ověřují, zda spolupracují správně s jinými moduly.
  + Testy komponent: testují celé funkční komponenty systému a ověřují, zda pracují správně při interakci s ostatními komponentami.
  + Integrační testy: testují interakce mezi různými komponentami nebo moduly a ověřují, zda spolupracují správně.
  + Funkční testy: testují, zda systém splňuje specifikované funkční požadavky.
  + Systémové testy: testují celý systém jako celek a ověřují, zda splňuje požadavky a funguje správně v prostředí, kde bude nasazen.
  + Akceptační testy: testují, zda systém splňuje požadavky uživatele a je připraven k uvedení do produkce.
* **Znalosti kódu**
  + White Box – tester zná interní implementaci části kódu která je testována, testuje danou implementaci
  + Black Box – tester nezná interní implementaci části kódu, která je testována testuje pouze externí design (nebo něco jako fasádu)
* **Způsobu realizace testů**
  + Manuální – musí existovat člověk který si sedne k aplikaci a celou si jí prokliká
  + Automatizované – někdo musí napsat testy které budou očekávat určitý výsledek po určité akci
  + Exploratory testing – tester prozkoumává aplikaci zkouší různé funkce a vlastnosti softwaru a snaží se naleznout nějaké chyby, které by jinak mohli být přehlédnuty

### Unit testování

Každý jednotkový test by měl testovat pouze jednu operaci jedné jednotky. Test by neměl ověřovat stav a interakci příliš mnoha nesouvisejících součástí (to by zvyšovalo závislost testu na konkrétní implementaci). Testy by se neměly vzájemně překrývat v tom, co testují.

V pythonu se unit testy dělají tak že se vytvoří samostatný modul, na importuje se unittest modul a vytvoří se třída která se dědí od unittest.TestCase následně se vytvoří metody s názvem test\_…. lze spustit celý modul a všechny testy v něm či po jednom testu.

**import** **unittest**

**class** **TestStringMethods**(unittest.TestCase):

**def** test\_upper(self):

self.assertEqual('foo'.upper(), 'FOO')

**def** test\_isupper(self):

self.assertTrue('FOO'.isupper())

self.assertFalse('Foo'.isupper())

**def** test\_split(self):

s = 'hello world'

self.assertEqual(s.split(), ['hello', 'world'])

*# check that s.split fails when the separator is not a string*

**with** self.assertRaises(**TypeError**) as error:

s.split(2)

self.assertEqual(str(error.exception),'must be str or None, not int')

**if** \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Dekorátory (píší se and funkci):

* @unittest.skip(reason)

Unconditionally skip the decorated test. reason should describe why the test is being skipped.

* @unittest.skipIf(condition, reason)

Skip the decorated test if condition is true.

* @unittest.skipUnless(condition, reason)

Skip the decorated test unless condition is true.

* @unittest.expectedFailure

Mark the test as an expected failure or error. If the test fails or errors in the test function itself (rather than in one of the test fixture methods) then it will be considered a success. If the test passes, it will be considered a failure.

| **Method** | **Checks that** | **New in** |
| --- | --- | --- |
| [assertEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertEqual) | a == b |  |
| [assertNotEqual(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotEqual) | a != b |  |
| [assertTrue(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertTrue) | bool(x) is True |  |
| [assertFalse(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertFalse) | bool(x) is False |  |
| [assertIs(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIs) | a is b | 3.1 |
| [assertIsNot(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNot) | a is not b | 3.1 |
| [assertIsNone(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNone) | x is None | 3.1 |
| [assertIsNotNone(x)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsNotNone) | x is not None | 3.1 |
| [assertIn(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIn) | a in b | 3.1 |
| [assertNotIn(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotIn) | a not in b | 3.1 |
| [assertIsInstance(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertIsInstance) | isinstance(a, b) | 3.2 |
| [assertNotIsInstance(a, b)](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertNotIsInstance) | not isinstance(a, b) | 3.2 |

## Dokumentace

Vnější – mimo zdrojový kód

* definice problému, požadavky, architektura
* popis rozhraní, tutoriál, FAQ
* popis návrhu, postupy, rozhodnutí, konvence
* uživatelská dokumentace

Vnitřní – součástí zdrojového kódu

* styl zápisu zdrojového kódu
* komentáře k souborům, třídám, funkcím, blokům
* popis rozhraní

Dokumentace v pythonu:

"""This is a module docstring.

This module contains functions for working with numbers.

"""

**class** **Person**():

"""Represents a person with attributes and methods."""

attr1: int

"""An integer representing the first attribute."""

attr2: int

"""An integer representing the second attribute."""

**def** **\_\_init\_\_**(**self**, attr3: int, attr4: int) -> None:

"""

Initializes a Person instance.

Args:

attr3 (int): An integer representing the third attribute.

attr4 (int): An integer representing the fourth attribute.

"""

self.attr3 = attr3

self.attr4 = attr4

**def** **my\_function**(**self**, arg1: int, arg2: int = 3) -> int:

"""

Performs a calculation and returns the result.

Args:

arg1 (int): An integer representing the first argument.

arg2 (int): An integer representing the second argument.

Defaults to 3.

Returns:

int: An integer representing the result of the calculation.

Raises:

Exception: If arg1 is 0.

"""

if arg1 == 0: #this is a really important comment

raise Exception("arg1 cannot be 0.")

return **self**.attr3 + arg2

Readme:

* **Název projektu** – uveďte název projektu a popište, co projekt dělá.
* **Popis projektu** – popište, co projekt dělá a jaké jsou jeho hlavní funkce. Vysvětlete také, proč jste projekt vytvořili a jak by mohl být užitečný pro ostatní.
* **Instalace** – popište, jak nainstalovat projekt a jaké jsou jeho závislosti. Uveďte také, jakým způsobem lze projekt spustit.
* **Nastavení projekt (setup projektu)**– jak a kde nastavit potřebné informace pro spuštění projektu, jaká možnost dělá, co a jaké parametry v ní mohou být a jaké naopak ne
* **Správa a práce s aplikací** – pokud ovládání projektu není úplně intuitivní tak popište, jak by se mělo s projekt zacházet, co jaká část dělá apod.
* **Příspěvky** – pokud byste chtěli, aby ostatní přispěli k projektu, uveďte, jakým způsobem to mohou udělat a jak by měli postupovat.
* **Kontaktní informace** – uveďte kontaktní informace, aby lidé mohli kontaktovat vás nebo tým, pokud mají dotazy nebo potřebují pomoc.
* **Licence** – pokud chcete projekt poskytnout pod licencí, uveďte, jaká licence se na projekt vztahuje.
* **Stav projektu** – uveďte, v jakém stavu se projekt nachází, například zda je aktivní, přestal se vyvíjet, nebo zda se jedná o prototyp.