Python a Robot Framework

Den druhý

ENGETO s.r.o.

Prezentované materiály jsou dostupné na adrese
https://github.com/ChaoticRoman/python-a-robot

Osnova kurzu

1. den: Filosofie, instalace a základy Pythonu, Python 2 a 3, standardní knihovna, objektově orientované programování, malé projekty s knihovnami numpy, matplotlib a Tkinter, distribuce aplikací pomocí setuptools a distutils

2. den: Středně velké projekty, organizace kódu, dokumentace, stanovení požadavků, teorie softwarového testování, unittesting, doporučené praktiky

3. den: Větší projekty, softwarové testování: integrační testování, systémové testování, akceptační testování, TDD, BDD, Robot Framework: základy

4. den: Kooperace pomocí gitu, logování, standardy, společný projekt,
Robot Framework: pokročilé funkce a doporučené praktiky

5. den: Testování a mocking, Robot Framework: uživatelská rozšíření, společný projekt

Den druhý

- Středně velké projekty
- Stanovení požadavků (Technical requirements specification)
- Organizace kódu
- Dokumentace
- Teorie softwarového testování
- Unittesting
- Doporučené praktiky

Stanovení požadavků (Technical requirements specification)

- Agile vs. Waterfall (BDUF): Jak najít rovnováhu?
  - Záleží jak drahá je chyba a jak se mění požadavky
- Vždy alespoň nastínit účel, požadované vlastnosti a rizika
- Až na ty nejmenší projekty, vždy aspoň odstavec
- Odsouhlasit zákazníkem
  - ne vždy to stačí, poslechněte si celý příběh využití

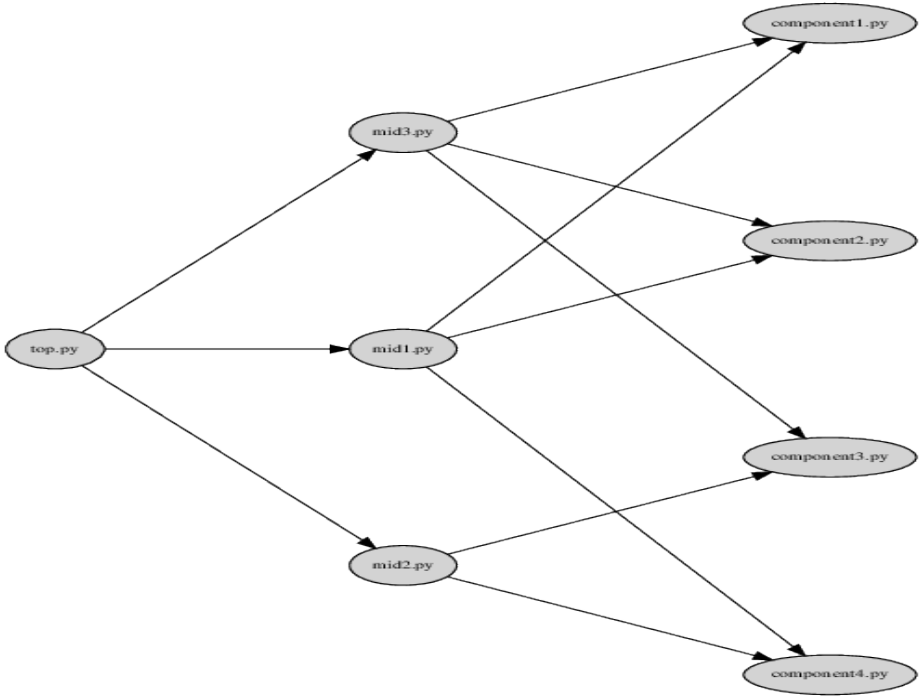
Středně velké projekty

- 2 až 10 kLOC (tisíců řádků kódu)
- Rozumná velikost jednoho modulu (.py souboru)
  - do 500 LOC
- Rozumné množství importovaných modulů na modul
  - do 5 vnitroprojektových importů na modul
- Rozumná hloubka hierarchie projektu
  - do 4 úrovní

Organizace kódu

- Single responsibility principle
  - Žádné misc.py, utils.py apod. :-)
- Hierarchie závislostí by měla být acyklický graf
- Třívrstvá organizace modulů
- Závislosti mají směřovat tak, aby technické detaily závisely na částech systému blízko problému zákazníka (tzv. business logika)
- snakefood pro vizualizaci organizace projektu

Třívrstvá organizace kódu: naivní a užitečný ideál



Dokumentace

- Chybná dokumentace může být horší než žádná
- Náročná dokumentace na údržbu → menší šanci být udržována
- Často se prostě zapomíná na reflexi změn v dokumentaci
- Řešení? Dokumentace v kódu + sphinx
- Odpovídejte na otázku “Proč?”, ne “Jak?”
- Kvalita, ne kvantita:
  - Když stačí jednostrákové README.md, mějte jen to
- Minimálně ovšem: účel, autoři a kontakt, závislosti, instalace, asumpce, použití, jak testovat

Teorie softwarového testování

- SW testování = podpora vývojářů
- Extra zdroje pro automatické testování → šetří zdroje později
- Nepříjemné testování → pravděpodobně špatný design
- Extenzivní testování vede k lepší arcitektuře
- Složení testů: nejvíce unittestů, daleko méně integračních testů, nejméně systémových testů (založeno na matematických základech)
- Google Testing Blog: Just Say No to More End-to-End Tests

Testing: Simple Testing Can Prevent Most Critical Failures

- A majority (77%) of the failures require more than one input event to manifest, but most of the failures (90%) require no more than 3.
- Almost all (98%) of the failures are guaranteed to manifest on no more than 3 nodes. 84% will manifest on no more than 2 nodes.
- 74% of the failures are deterministic — they are guaranteed to manifest given the right input event sequences.
- A majority of the production failures (77%) can be reproduced by a unit test.
- Almost all catastrophic failures (92%) are the result of incorrect handling of non-fatal errors explicitly signaled in software.
- 35% of the catastrophic failures are caused by trivial mistakes in error handling logic — ones that simply violate best programming practices and that can be detected without system specific knowledge.

From: Yuan et al. (2014), Simple Testing Can Prevent Most Critical Failures

Unittesty

- Testujte nejen “happy” scénaře, ale i ty negativní
- Testujte každou funkci, která obsahuje logiku
- Separace odpovědností → méně scénářů → snadný unittesting
- Extenzivní závislost na komponentech třetích stran → Oddělte logiku a volání těchto komponent
- Mocking nedostupných komponent
- Kontrolujte výsledky, nejen průběh bez zjevné havárie
- Nechte si testy jednou za čas záměrně selhat, ať vidíte, že testují, co testovat mají

Hands-on: Free-style projekt

- Co byste rádi?
- Inspirace
  - Vizualizace
  - Pracovní nástroj nebo úkol
  - …
- Začněte odstavcem s popisem požadavků a rizik

Den druhý: rekapitualce

- Středně velké projekty
- Organizace kódu
- Dokumentace
- Stanovení požadavků (Technical requirements specification)
- Teorie softwarového testování
- Unittesting