MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Nástroje na analýzu kódu v jazyku Python a vizualizácia ich výstupu

BAKALÁRSKA PRÁCA

Ján Vorčák

Brno, 2012

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Ján Vorčák

Vedúci práce: Mgr. Marek Grác

Poďakovanie

Zhrnutie

Kľúčové slová

Obsah

1 Úvod

Pri vývoji aplikácií musíme klásť dôraz nielen na samotné programovanie, ale najmä na analýzu a návrh systému, dôkladné otestovanie, ale aj spätnú analýzu samotného kódu. Z tohto dôvodu vzniká pre takmer každý programovací či značkovací jazyk množstvo nástrojov, ktoré nám pomáhajú automaticky objaviť potencionálne chyby či porušenie konvencií v zdrojových kódoch. Častokrát sú tieto nástroje síce efektívne a ľahko konfigurovateľné, no najmä výstupom nie príliš užívateľsky prívetivé. Asi najpoužívanejším nástrojom na analýzu kódu v jazyku Python je konzolová utilita Pylint. Pokiaľ si chceme vďaka tomuto programu vytvoriť obraz o väčšom projekte, nestačí nám iba textový výstup, no je potrebné tieto dáta automaticky spracovať.

Cieľom tejto bakalárskej práce je analyzovať nástroje na analýzu a vizualizáciu Python kódu ako aj vlastnosti jazyka, ako je napríklad introspekcia, ktoré nám túto analýzu umožnujú. Z dôvodu vizualizácie je nutné taktiež naštudovať základy modelovania hierarchických štruktúr v UML. Výstupom bakalárskej práce je nástroj, ktorý poskytuje funkcionalitu, ktorú v existujúcich nástrojoch nenájdeme. Nástroj teda zjednoduší orientáciu najmä vo väčšom projekte pomocou vizualizácie dát najmä za pomoci vygenerovania interaktívnych UML diagramov. Program bude taktiež dopĺňať funkcionalitu, ktorú nám klasický program Pylint neumožnuje, ako je napríklad filtrácia falošných poplachov.

Práca pozostáva z piatich kapitol. V druhej kapitole si predstavíme jazyk Python a jeho vlastnosti, neskôr rozanalyzujeme existujúce nástroje na analýzu kódu v jazyku Python a ich nedostatky. Pred implementačnou časťou si predstavíme knižnice, ktoré nám neskôr pomôžu pri implementácií, nástroja ako je napríklad knižnica Gaphas na vykresľovanie grafiky v GTK+ programoch alebo samotný Pylint. Nasledovať bude kapitola o návrhu nástroja a samotná implementácia. Nakoniec zhodnotíme, či implementovaný nástroj spĺňa určené požiadavky či už po stránke funkcionality, alebo škálovateľnosti a navrhneme zmeny na jeho zlepšenie.

2 Python

2.1 Charekteristika jazyka Python

Python je vysoko úrovňový, objektovo orientovaný dynamický jazyk, ktorý vytvoril holandský programátor Guido van Rossum ako následníka jazyka ABC. Vyznačuje sa prehľadnou syntaxou, jednoduchosťou a modulárnosťou. Python podporuje viacero programátorských paradigiem, najmä objektovo orientované, imperatívne a čiastočne aj funkcionálne paradigmy.

Existuje viacero implementácií jazka Python. Medzi najznámejšie patria:

- CPython
- Jython
- Python pre .NET
- IronPython
- PyPy

Najpožívanejšou a najpodporovanejšou je ale CPython. Každá z implementácií sa môže líšiť špecifickými informáciami mimo štandardnej Python dokumentácie. Zdrojové kódy sú preložené do byte kódu a zvyčajne uložené v .pyc alebo .pyo súboroch, ktoré sú neskôr spúštané virtuálnym strojom.

V štandardnej knižnici je dostupné množstvo dátových typov, ako napríklad reálne a komplexné čísla, celé čísla s neobmedzenou dĺžkou, znakové reťazce, zoznamy a slovníky. Dátové typy sú silno a dynamicky typované. Použitie nekompatibilného typu spôsobí vyvolanie výnimky. Python podporuje objektovo orientované programovanie vrátane viacnásobnej dedičnosti. Kód je sústredený do modulov a balíkov s možnosťou importovať špecifický modul, triedu, funkciu alebo iný objekt. Za účelom ošetrenia chýb Python podporuje vyvolávanie a odchytávanie výnimiek. Automatická správa pamäti nahrádza nutnosť manuálne alokovať a uvoľnovať pamäť v kóde.

2.2 Introspekcia v jazyku Python

Introspekcia je schopnosť preskúmať daný objekt a rozhodnúť o jeho identite, vlastnostiach a schopnostiach. Jazyk Python podporuje rozsiahlu introspekciu objektov. Medzi hlavné informácie, ktoré potrebujeme o objektoch v jazyku Python zistiť, patrí ich meno, typ, identita, vlastnosti, schopnosti a

ich pôvod. Jazyk Python ponúka množstvo nástrojov, ktoré nám tieto vlastnosti umožnujú zistiť. Môžeme ich rozdeliť do dvoch hlavných skupín. V prvej sú funkcie či už zo štandardnej knižnice, alebo z pomocných modulov, akým je napríklad modul inspect. Do druhej skupiny radíme atribúty objektov, ktoré priamo v objekte uchovávajú užitočné informácie.

- Funkcia dir() je jedným z hlavných nástrojov introspekcie v jazyku Python a vracia zotriedený zoznam mien atribútov objektu, ktorý bol uvedený ako jej argument. Funkcia je súčasťou štandardnej knižnice, takže nemusíme importovať žiaden modul pre jej použitie. Na výpis funkcií zo štandardnej knižnice môžeme teda využiť samotnú funkciu dir().

  ```
  In [1]: print dir(__builtin__)[-10:]
  ['str', 'sum', 'super', 'tuple', 'type', 'unichr',
      'unicode', 'vars', 'xrange', 'zip']
  ```

  V prípade, že je funkcia dir() použitá bez argumentov, vracia zoznam mien, ktoré sú momentálne definované.

  ```
  In [2]: print dir()
  ['In', 'Out', '_', '__', '___', '__builtin__', '
     __builtins__', '__name__', '_dh', '_i', '_i1',
     '_i2', '_ih', '_ii', '_iii', '_oh', '_sh', '
     exit', 'get_ipython', 'help', 'quit']
  ```

- Atribút __doc__ obsahuje komentáre, ktoré popisujú objekt. V prípade, že prvý v module, triede alebo v metóde je znakový reťazec, je automaticky považovaný za __doc__ atribút. V opačnom prípade sa jeho hodnota nastaví na None. Hodnoty __doc__ sa z dôvodu kompaktnosti nevkladajú do byte kódu.

  ```
  In [3]: print __doc__.__doc__
  str(object) -> string

  Return a nice string representation of the object.
  If the argument is a string, the return value is
     the same object.
  ```

- Atribút __name__ obsahuje názov objektu odvodený z jeho typu. Niektoré objekty, ako napríklad objekty typu string tento atribút neobsahujú. Tento atribút obsahujú napríklad moduly. V prípade, že spúšťame

skript priamo pomocou Python interpretu, je atribút __name__ nastavený na hodnotu '__main__', nakoľko Python interpret je považovaný za hlavný modul a to aj v prípade, že Python skript spúštame z príkazového riadku. Je teda často používaný na rozpoznanie, či daný modul len importujeme, alebo priamo spúštame.

```
In [4]: def my_function():
   ....:     pass
   ....:

In [5]: my_function.__name__
Out[5]: 'my_function'

In [6]: __name__
Out[6]: '__main__'
```

- Funkcia type() zo štandardnej knižnice vracia typ jej argumentu. Ten vracia v podobe typového objektu, ktorý môže byť porovnávaný s typmi definovanými v module types.

```
In [7]: type(my_function)
Out[7]: function

In [8]: type(1)
Out[8]: int
```

- Funkcia id() vracia unikátnu identitu objektu. Táto funkcia je užitočná, nakoľko viacero premenných môže odkazovať na rovnaký objekt. Funkcia id() konkrétne vracia pamäťovú adresu objektu.

```
In [9]: id('string')
Out[9]: 3078023712L

In [10]: id(id)
Out[10]: 3078187660L
```

- Funkcie hasattr() a getattr() - v prípade, že je potrebné zistiť prítomnosť alebo hodnotu atribútu, štandardná knižnica ponúka funkcie hasattr() a getattr().

```
In [11]: hasattr(id, '__name__')
Out[11]: True
```

```
In [12]: getattr(id, '__name__')
Out[12]: 'id'
```

- Funkcia callable() - v niektorých prípadoch môžu objekty slúžiť na vyvolanie určitého druhu udalostí. Pomocou funkcie callable() sa dá overiť, či je daný objekt spustiteľný.

```
In [13]: callable(id)
Out[13]: True

In [14]: callable(1)
Out[14]: False
```

- Funkcie isinstance() a issubclass() - funkcia isinstance() vracia hodnotu v závislosti od toho, či je objekt inštanciou danej triedy. Funkcia vracia pravdivú hodnotu aj v prípade, že je objekt inštanciou jej predka.

  Funkcia issubclass() vracia pravdivú hodnotu v prípade, že objekt reprezentujúci triedu je podtriedou druhého argumentu.

```
class Person(object):
    pass
class Student(Person):
    pass
p=Person()
s=Student()

In [15]: isinstance(s, Student)
Out[15]: True

In [16]: isinstance(s, Person)
Out[16]: True

In [17]: issubclass(Student, Person)
Out[17]: True
```

3 Nástroje na analýzu kódu pre jazyk Python

3.1 Nástroje analýzu kódu Python projektu

3.1.1 PEP8

Ide o nástroj, ktorý vyvíja Johann C. Rocholl. Tento nástroj slúži na otestovanie kódu Python podľa konvencií definovaných v PEP 8 [**?**]. Je to konzolový program, ktorý pomáha zvyšovať prehľadnosť a lepšiu štruktúru kódu.

Program PEP8 umožnuje programátorovi pomocou rôznych prepínačov ovládať analyzátor. V predvolenom móde vypíše program každú chybu len raz, čo je pre analýzu rozsiahlejšieho projektu nevhodné. Voľba `-repeat` vypíše každú chybu bez ohľadu na to, či sa už v kóde vyskytuje viackrát alebo nie. Voľba `-filename=patterns` obmedzuje spracovávané súbory len na tie, ktoré vyhovujú zadanému regulárnemu výrazu. Zaujímavým prepínačom je `-show-pep8`, ktorý k danej chybe vypíše na štandardný výstup aj text z definície PEP 8, ktorý daná časť kódu porušuje. Utilita Pep8.py je nástroj, ktorý výrazne dopomáha k prehľadnosti a ucelenosti kódu. Chýbajú mu však možnosti detekovania chýb v kóde [**?**].

3.1.2 PyChecker

PyChecker je program zachytávajúci problémy, na ktoré upozorňuje u nedynamických jazykov, ako je napríklad C kompilátor v podobe varovaní a chýb. Medzi problémy, ktoré tento program zachytí, patrí napríklad zlý počet parametrov predaný funkcii, metóde či konštruktoru, používanie neexistujúcich metód a tried, ako aj použitie premennej pred jej inicializáciou. Detekuje taktiež nepoužité inštancie globálnych či lokálnych premenných, kontroluje definíciu `self` ako prvej premennej u metód tried, ako aj úroveň dokumentácie pre triedy, moduly a metódy [**?**].

Jednou z predností je aj priamy import do zdrojového kódu. Pokiaľ má užívateľ prístup a práva k modifikácii zdrojového kódu, je veľmi jednoduché PyChecker naimportovať priamo. Jeho hlavnou nevýhodou je, že daný kód spustí a vykoná. PyChecker je teda nevhodný na detekciu chýb v aplikáciách, ktoré napríklad pracujú nad databázou, alebo pracujú v ostrom prostredí, ktoré je nevhodné na testovanie.

3.1.3 Pylint

Pylint je jeden z najpoužívanejších, ľahko konfigurovateľných nástrojov na analýzu kódu Python, využívaný či už manuálne, alebo automaticky. Je vyvíjaný za podpory Logilab.org a vydaný pod licenciou GNU GPL [**?**]. V zdrojových kódoch analyzuje potencionálne chyby a varovania, ale aj porušenie konvencií podľa štandardu PEP 8. S projektom Pylint je dodávaný aj program Pyreverse, ktorý pre daný kód vygeneruje UML diagram v podobe *dot* [**?**] formátu. Daný *dot* súbor dokáže priamo pretransformovať do *PNG* alebo iného formátu. Výhodou programu Pylint oproti PyChecker je, že daný súbor väčšinou nespúšta, ale analyzuje staticky. Je teda vhodný do každého prostredia. Pylint dopĺňa PyChecker hlavne v kontrole dĺžky riadkov a kontrole názvov premenných za pomoci regulárnych výrazov. Overuje taktiež implementáciu deklarovaných rozhraní a detekuje duplicitný kód. Obzvlášť zaujímavou vlastnosťou je generovanie metrík a externých závislostí [**?**].

Správanie sa programu je možné upraviť pomocou prepínačov špecifikovaných na príkazovom riadku alebo pomocou konfiguračného súboru. V konfiguračnom súbore je možná filtrácia varovaní a chýb rôznych druhov, ignorovanie rôznych typov súborov. Taktiež je možné načítať rôzne typy doplnkov, nastavenie minimálneho počtu znakov v riadkoch, obmedzenie veľkosti modulu, nastavanie znakov využitých na odsadenie, vlastné definície zastaralých modulov.

Najmä u väčších projektov s požiadavkami na prehľadnosť kódu a dobrý návrh užívateľ ocení možnosť nakonfigurovať maximálny počet atribútov triedy pomocou prepínača *max-attributes*, rodičov triedy pomocou prepínača *max-parents*. Počet výrazov *return* a *yield* je možné obmedziť s využitím prepínača *max-returns*. U niektorých existuje možnosť špecifikovať aj minimálny počet, napríklad u počtu verejných metód triedy pomocou prepínačov *min-public-methods* alebo *max-public-methods* [**?**].

3.1.4 Zhrnutie

Pylint patrí medzi najpoužívanejší program na analýzu kódu Python, nakoľko je ľahko konfigurovateľný a kombinuje funkcionalitu ostatných nástrojov. Do každého prostredia je vhodné vybrať nástroj na analýzu vždy unikátne. Aj keď Pylint môžeme nazvať najpoužívanejším nástrojom, projekty ako Moap, Savon alebo Flumotion využívajú pre testovanie PyChecker.

Program PEP8 má taktiež široké použitie, no len s obmedzením na porušenie konvenčných chýb, ktoré sú v niektorých prípadoch akceptovateľné,

hlavne za účelom sprehľadnenia kódu alebo z historických dôvodov projektu.

3.2 Nástroje na vizualizáciu Python projektu

Nástroje na vizualizáciu nám slúžia na vygenerovanie UML diagramov ale aj na zobrazenie dodatočných informácií o projekte, akými sú napríklad metriky alebo chyby v kóde. V tejto kapitole si predstavíme dostupné nástroje na vizualizáciu Python kódu.

3.2.1 Pyreverse

Pyreverse je program dodávaný spoločne s utilitou Pylint, jeho úlohou je vygenerovať UML diagramy pre Python projekt. Vykonáva synaktickú analýzu Python balíkov a vygeneruje UML diagram vo formátoch dot alebo vgc. Pyreverse je za pomoci programu dot schopný vygenerovať aj UML diagram vo formáte PNG.

Medzi hlavné výhody programu Pyreverse patrí jeho rýchlosť a široká ponuka prepínačov, ktoré umožňujú ignorovať moduly a súbory, filtrovať typy atribútov alebo ovládať výzor výsledného UML diagramu. Zaujímavou vlastnosťou je vygenerovanie diagramu tried súvisiacich s jednou konkrétnou triedou. Túto vlastnosť umožnuje prepínač –class=<class>.

Nevýhodou programu Pyreverse je jeho neinteraktívnosť. Pri editácii projektu musíme daný graf pregenerovať. Navyše nemôžeme pomocou programu Pyreverse editovať projekt.

```
$ pyreverse -o png --ignore=gaphas .
```

3.2.2 Pylint-gui

Pylint-gui je program, ktorý bol vytvorený najmä pre užívateľov operačného systému Windows ako alternatíva k spúštaniu Pylint pomocou príkazového riadka. Grafické rozhranie programu využívajúce Tk je veľmi jednoduché a jedná sa len o prevedenie konzolového výstupu do vizuálnej podoby. Pylint-gui teda neodstránil základné nedostatky programu Pylint. Pri editácii projektu treba výstup znova pregenerovať, nakoľko chyby sú identifikované číslom riadku, ktorý sa pri editácii môže zmeniť.

3.2.3 Implementácia programu Pylint v IDE

Program Pylint bol implementovaný do väčšiny IDE prostredí, ktoré ho graficky a dynamicky interpretujú. Táto implementácia je často len v podobe doplnkov a modulov bez priamej podpory vývojárov IDE. Ďalšou z nevýhod je veľká robustnosť IDE prostredí a obmedzenosť na dané vývojové prostredie.

Jedným z príkladov týchto doplnkov je modul PyDev pre vývojové prostredie Eclipse. Jeho výhodou je jeho automatické spustenie po zmene súboru, ktoré však ale pri väčších projektoch môže spôsobiť spomalenie. Výhodou je aj možnosť prefiltrovať typ chyby, ktorú Pylint zaznamená, obmedzuje sa ale len na nasledujúcich päť skupín chýb, nie však na konkrétne chybové správy.

- FATAL
- ERRORS
- WARNINGS
- CONVENTIONS
- REFACTOR

Nepodporuje taktiež filtráciu falošných poplachov ani generáciu UML.

3.2.4 Zhrnutie

Existuje množstvo utilít a programov, ktoré pomáhajú výsledky analýzy Python kódu vizualizovať. Okrem vyššie uvedených môžeme spomenúť aj PyCAna, Pymetrics alebo Pypants. Medzi základné problémy, ktoré tieto programy majú patrí ich neinteraktívnosť, nutnosť výsledky pregenerovať po editácii kódu, robustnosť alebo chýbajúca funkcionalita. Pri implementácii projektu sa pokúsime zamerať práve na tieto chýbajúce vlastnosti. Výsledný program by mal byť rýchly a neobmedzovať programátora na jeden druh editoru, ako je to napríklad pri doplnkoch IDE.

4 Analýza nástrojov potrebných k implementácii

4.1 Analýza nástroja Gaphas

4.1.1 Základná charakteristika nástroja Gaphas

Gaphas predstavuje zoskupenie knižníc a nástrojov na vykresľovanie grafických objektov na určené elementy grafického rozhrania GTK. Je naprogramovaný v jazyku Python a vydaný pod ??? licenciou.

4.1.2 Popis Gaphas API

Gaphas API využíva MVC návrhový vzor a môžeme ho teda rozdeliť na 3 hlavné časti - model, view a controller.

- Model - je časť API, ktorá obsahuje:
    - api/canvas - plátno na vykresľovanie, ktoré sa správa ako kontajner pre vykresľované položky. Canvas v sebe zahŕňa atribúty, ktoré súvisia s vykresľovaním, napríklad atribút gaphas.Solver alebo atribút __connections uchovávajúci informácie o prepojeniach medzi jednotlivými položkami
    - api/items - položky gaphas.item.Element a gaphas.item.Line odvodené od triedy gaphas.item.Item, ktoré sú uchovávané a vykresľované na plátne Jednoduchým odvodením novej triedy z gaphas.item.Item môžeme vytvoriť vlastný prvok na vykresľovanie
    - api/connectors - v tejto kategórií sa jedná o triedy gaphas.connector.Handle a gaphas.connector.Port, ktoré umožnujú spájať jednotlivé položky na plátne
    - api/solver - umožnuje definovať podmienku medzi aspoň dvomi premennými a a udržovať túto podmienku ako pravdivú pri zmene premenných.
    - api/constraint - obmedzenia pre premenné, ktoré sú zaregistrované na plátne. Každé obmedzenie obsahuje zoznam premenných, ktoré sú zaregistrované v objekte typu gaphas.solver.Solver
    - api/utils - obsahuje pomocné funkcie týkajúce sa najmä vykresľovania textu na plátno

- View - obsahuje všetky triedy súvisiace so zobrazovaním a vykresľovaním jednotlivých elementov:
  - api/view - trieda, ktorej úlohou je spravovať vykresľovanie Gaphas položiek, uchováva vykresľovacie plátno, informácie o označených objektoch a objektoch, nad ktorými sa nachádza myš
  - api/painters - objekty, ktoré vykresľujú jednotlivé objekty
  - api/gtkview - trieda implementujúca funkcionalitu gaphas.view.View v Gtk.DrawingArea. Slúži teda na vykreslenie plátna na Gtk prvok. Používa nástroje z časti controller, a objekty z časti view
- Controller - časť API slúžiaca na interakciu s plátnom a objektami obsahuje:
  - api/tools - nástroje ktoré poskytujú pohľadu interaktívnosť tým, že spracovávajú udalosti ktoré sú ním posielané. Gaphas API poskytuje HoverTool slúžiaci k označeniu položky, ktorá sa nachádza pod myšou, ItemTool poskytuje výber a premiestňovanie položiek, HandleTool výber a pohyb s objektami typu gaphas.connector.Handle. Okrem nich ponúka aj nástroje ako PanTool pre pohyb plátna alebo PlacementTool pre umiestnenie nových položiek na plátno. Nástroje môžu byť nakombinované a zreťazené do jedného nástroja kombinujúceho ich vlastnosti s použitím triedy ToolChain. Nástroje sú implementované pomocou udalostí. Nástroj môže udalosť obslúžiť alebo ignorovať. Existuje teda jednoduchá možnosť v prípade potreby naimplementovať vlastný nástroj.
  - api/aspects - definujú funkcionalitu na rozmedzí položiek a nástrojov. Vysporiadajú sa teda s pohybom položiek alebo ich označením

4.1.3 Zhrnutie

Gaphas knižnica umožnuje vykresľovanie grafiky na plátno v prostredí GTK+. Jej výhodou je možnosť prispôsobiť vykresľované objekty priamo na mieru. Výborne spracovaná je aj interakcia medzi objektami. Vďaka constraints môžeme dva objekty udržovať v definovanej pozícii, napríklad na jednej čiare, v jednom bode alebo na pozícii definovanou rovnicou.

4.2 Analýza nástroja Pylint

4.2.1 Pylint

Kľúčová trieda v štruktúre programu Pylint je trieda pylint.lint.Pylinter, ktorá spravuje nastavenia, doplnky, aktiváciu a deaktiváciu správ na úrovni modulov. Uchováva taktiež informácie o počte tried, metód a iné jednoduché štatistiky. Táto trieda je spúšťaná hlavnou triedou Run pri spustení programu. Spravuje taktiež checkers, ktoré analyzujú kód a reporter - triedu zopovednú za výstup.

V nasledujúcom odseku si podrobne opíšeme najdôležitejšie prvky programu Pylint.

- Checkers

  Ide o triedy, ktoré implementujú aspoň jednu z tried

  - IRawChecker
  - IASTNGChecker

  Tieto triedy sú zaregistrované v triede pylint.lint.Pylinter pomocou jej metódy register_checker. Po zaregistrovaní sa prevedie hlavná práca v metóde pylint.lint.Pylinter.check, kde sa rozdelia zaregistrované triedy podľa ich predkov a začne sa prehľadávanie a kontrola modulov. V triedach checker sa zavolá metóda process_module, ktorej implementácia je ponechaná na každej triede samostatne.

  Každá takáto trieda má v zaregistrovaný slovník chybových správ ktorý dokáže detekovať. Ako kľúč je použitý kód chyby a ako hodnota dvojica obsahujúca textovú reprezentáciu chyby a jej popis.

```
MSGS = {
    'W0511': ('%s',
              'Used when a warning note as FIXME
                 or XXX is detected.'),
    }
```

  Slovník chybových hlášok triedy EncodingChecker.

  Pri náleze chyby checker zaregistruje chybu pomocou metódy add_message zdedenej z triedy pylint.checkers.BaseChecker, ktorá chybu predá instancii triedy pylint.lint.Pylinter.

```
def add_message(self, msg_id, line=None, node=
    None, args=None):
     """add a message of a given type"""
     self.linter.add_message(msg_id, line, node
         , args)
```

- Reporters Tento typ tried je odvodený od triedy pylint.reporters.BaseReporter. Triedy slúžia na formátovanie a vygenerovanie samotného výstupu. Pri inicializácií sa triede pomocou metódy set_output nastaví výstupný prúd, ktorý je v predvolenom režime nastavený na štandardný výstup. Je ale možné ho predefinovať na súbor alebo iný typ výstupného prúdu. Tak ako pri triedach typu checker je aj v tomto prípade veľmi jednoduché vytvoriť vlastnú implementáciu.

  V programe Pylint ale nájdeme pomerne široký výber výstupov.

  - HTMLReporter
  - GUIReporter
  - TextReporter
  - ParseableTextReporter
  - VSTextReporter
  - ColorizedTextReporter

4.2.2 Pyreverse

Pri prehľadávaní projektu si program Pyreverse vygeneruje s daných argumentov instanciu triedy logilab.astng.manager.Project za pomoci triedy ASTNGManager z balíčka logilab.astng, ktorý je závislosťou projektu Pylint. Táto trieda obsahuje informácie o názve, ceste, moduloch a premenných.

Za pomoci triedy logilab.astng.Linker preskenuje daný projekt a vygeneruje vzťahy medzi jednotlivými triedami.

Inštancie týchto dvoch tried sú predané ako parametry objektu typu DiadefsHandler, ktorý za pomoci triedy ClassDiadefGenerator vygeneruje diagram tried. Diagram tried je neskôr predaný triede zodpovednej za zápis.

Nasledujúca časť kódu je hlavnou kostrou programu Pyreverse.

```
try:
    project = self.manager.project_from_files(args)
    linker = Linker(project, tag=True)
    handler = DiadefsHandler(self.config)
```

```
    diadefs = handler.get_diadefs(project, linker)
finally:
    sys.path.pop(0)

if self.config.output_format == "vcg":
        writer.VCGWriter(self.config).write(diadefs)
else:
    writer.DotWriter(self.config).write(diadefs)
```

Triedou zodpovednú za zápis rozumieme triedu, ktorá rozširuje triedu pylint.pyreverse.writer.DiagramWriter.

Príkladmi takýchto tried sú napríklad nasledujúce triedy:

- DotWriter
- VCGWriter

Úlohou týchto tried je implementovať medódu set_printer, ktorá nastaví backend, ktorý bude zapisovať do súboru. Implementujú aj medódy get_title get_values a close_graph. Prácu za nich vykonáva funkcia write, ktorú zdedia po triede DiagramWriter, ktorá pomocou interných metód write_packages a write_classes vyvoláva metódy backendu pre zápis. Pri implementácií novej triedy typu writer je teda dôležité naimplementovať triedu, ktorá sa využíva ako backend a inicializovať a nastaviť ju vo vnútri metódy set_printer. Dôležité je taktiež backend triedu korektne uzavrieť v metóde close_graph, ktorá je taktiež volaná metódou write.

5 Implementácia nástroja

5.1 Cieľ

Mojím cieľom je naimplementovať nástroj, ktorý bude schopný prehľadne a interaktívne zobrazovať chyby v kóde v rámci hierarchie modulov, tried a funkcií a taktiež ich zobraziť v zdrojovom kóde bez nutnosti pregenerovania. Nástroj bude zobrazovať zdrojové kódy a umožnovať ich editáciu, pričom nebude obmedzený na jeden typ editoru. Taktiež by mal fungovať v plnej funkčnosti multiplatformne.

5.2 Zdôvodnenie výberu knižníc a nástrojov pre implementáciu

Program bude implementovaný ako GUI aplikácia v programovacom jazyku Python, nakoľko vďaka jeho introspekcii je väčšina programov a knižníc pre jeho analýzu naprogramovaná práve v ňom.

Ako grafické rozhranie som zvolil GTK+ toolkit, nakoľko spĺňa moje požiadavky a tými sú multiplatformnosť, kompaktnosť a rýchlosť. Aj napriek tomu, že bol pôvodne vyvíjaný pre X Window System je možné ho implementovať aj pre platformu Microsoft Windows a Quartz.

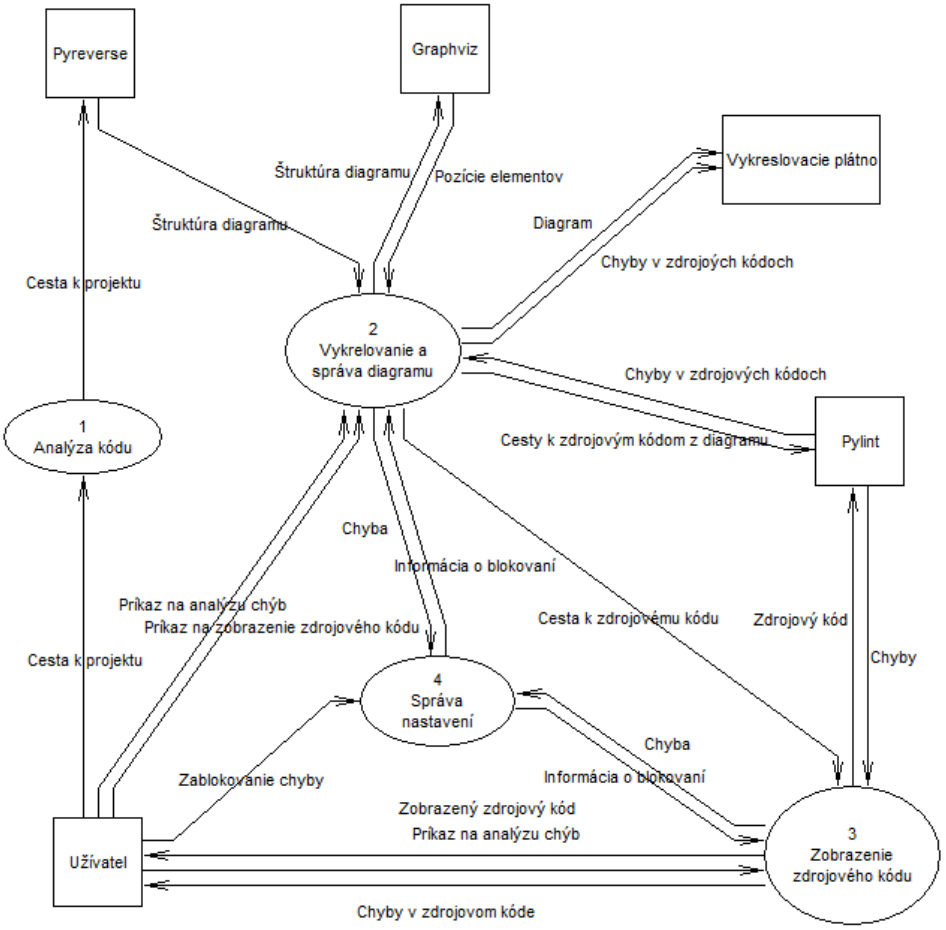
Backend systému bude zabezpečovať program Pylint, nakoľko je dostupný na všetkých platformách a je dlhodobo používaný a otestovaný. Fakt že je implementovaný do viacerých populárnych IDE potvrdzuje prítomnosť výborne navrhnutej API. Okrem generovania chýb poskytuje taktiež služby Pyreverse, vďaka ktorým bude možné naimplementovať generovanie diagramu tried a modulov.

Nakoľko bude ako grafické prostredie použitá knižnica GTK+ je jednoduché použiť ako editor GtkSourceView komponentu, ktorú ponúka táto knižnica. Nakoľko ale nechcem obmedziť program len na jeden druh editoru vytvorím jednoduché rozhranie pomocou ktorého naimplementujem napríklad editor Vim a umožním implementovať ostatné editory v prípade potreby. V prípade editorov je nutné aby boli schopné zobrazovať na chybových miestach príslušné značky. V prípade GtkSourceView editoru využijeme jeho funkcionalitu značiek. V prípade editoru Vim využijeme možnosť zvýraznenia určitých riadkov.

Pre zobrazenie grafu modulov a tried využijeme knižnicu Gaphas dostupnú pre GTK+. Knižnica nám pomôže vizualizovať výsledky analýzy. Jej jediným nedostatkom je neprítomnosť prvkov pre UML. Prvky pre zobrazenie tried, modulov a funkcií bude nutné doprogramovať manuálne.

5.3 Analýza a návrh

Nakoľko program využíva vnútornú štruktúru viacerých programov a knižníc, úlohou anlýzy je vhodne navrhnúť spôsob ich využitia. Nasledujúci DFD diagram popisuje základný tok dát v aplikácií.



Pri spustení programu užívateľ zvolí cestu k projektu pomocou grafického rozhrania alebo ako parameter na príkazovom riadku. Ten sa predá triede, ktorá používa metódy z knižnice Pyreverse. Vygenerovaná štruktúra reprezentujúca diagram sa predá knižnici Graphviz, ktorá rozhodne o pozíciách jednotlivých elementov diagramu tak aby bol diagram čo najprehľadnejší. Pre každý element v štruktúre diagramu sa vytvorí objekt knižnice Gaphas, ktorý sa vykreslí na plátne. Referencia na diagram a položky na plátne je

uložená ako kontext plátna pre neskoršie využitie. Pri spustení analýzy kódu projektu sa pomocou Pylint knižnice vygenerujú chybové hlásenia, prefiltrujú sa tak aby sa užívateľovi nezobrazovali hlásenia, ktoré v minulosti zablokoval. Následne pomocou kontextu plátna predajú informácie o počte nájdených chýb a informáciách o nich priamo jednotlivým položkám na plátne a zabezpečia ich prekreslenie. Tieto položky si uchovávajú napríklad aj aj informácie o pozícií v súbore. Po kliknutí na položku plátna sa vytvorí nové okno s editorom podľa aktuálnych nastavení. Pri kontrole zdrojového kódu danej triedy sa pošle požiadavka danému editoru, ktorý sa stará o zobrazovanie chýb.

5.4 Aplikácia jednotlivých nástrojov

Pre požiadavky aplikácie sme potrebovali vytvoriť triedy využívajúce alebo upravujúce funkcionalitu jednotlivých nástrojov. V nasledujúcej podkapitole popíšeme využitie jednotlivých knižních a ich implementáciu v našom projekte.

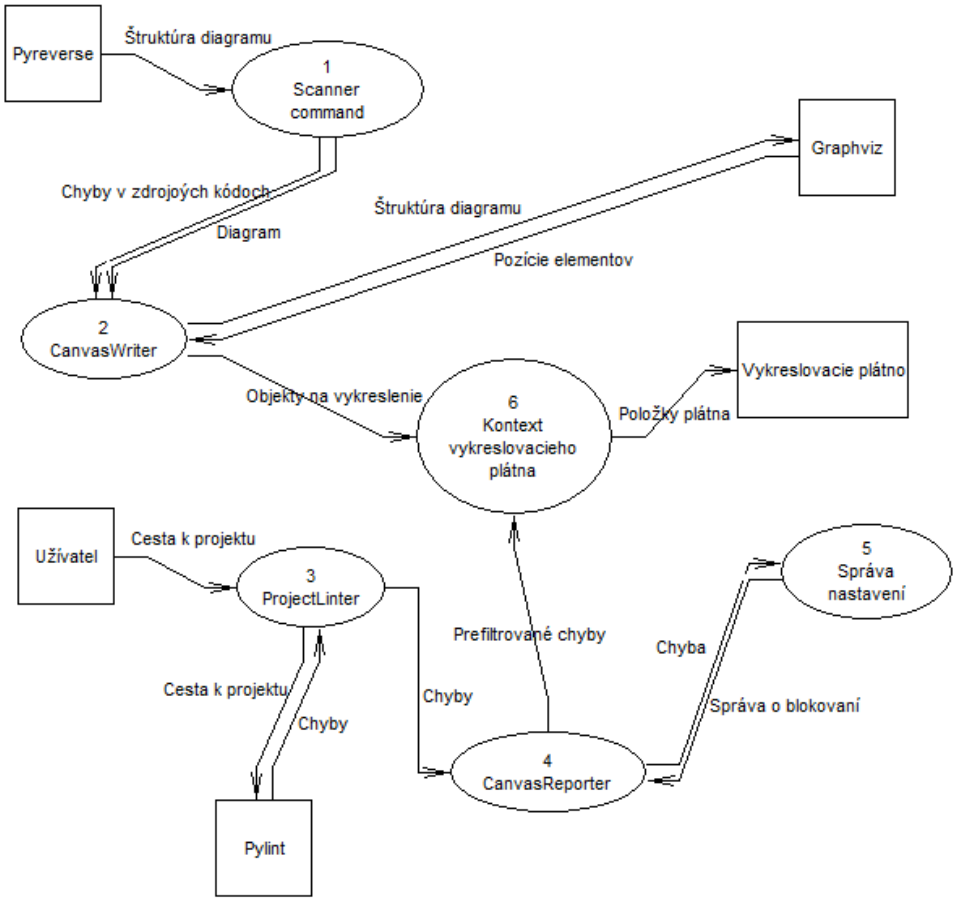
5.4.1 Aplikácia nástroja Pyreverse

Pri popise nástroja Pyreverse sme poukázali na hlavnú kostru programu Pyreverse, ktorá vracia diagram tried, predávaný triede zodpovednej za zápis. Vytvorili sme teda triedu ScannerCommand, ktorá spúšťa kostru programu Pyreverse v samostatnom vlákne, prefiltruje výsledky pre potreby aplikácie a výsledky predá zapisujúcej triede, ktorú naimplementujeme.

Triedu zodpovednú za zápis na plátno sme nazvali CanvasWriter. Táto trieda implementuje metódu get_values(self, obj), ktorá rozhoduje aké informácie budú predané jednotlivýcm položkám na plátne. V tejto metóde teda rozhodneme, ktoré údaje od programu Pyreverse predáme k dalšiemu spracovaniu. Inak len deleguje funkcionalitu na triedu CanvasBackend.

Trieda CanvasBackend rozširuje triedu DotBackend z programu Pylint. Všetky výstupy tak generuje do dot formátu, ktoré v prekrytej metóde generate(self, filename) použijeme ako vstup pre knižnicu Gaphas.

Vďaka nej dostaneme pozície jednotlivých elementov na plátne tak aby bol výsledný graf prehľadný a uložíme potrebné informácie do kontextu plátna. Daný graf následne vykreslíme na plátno.



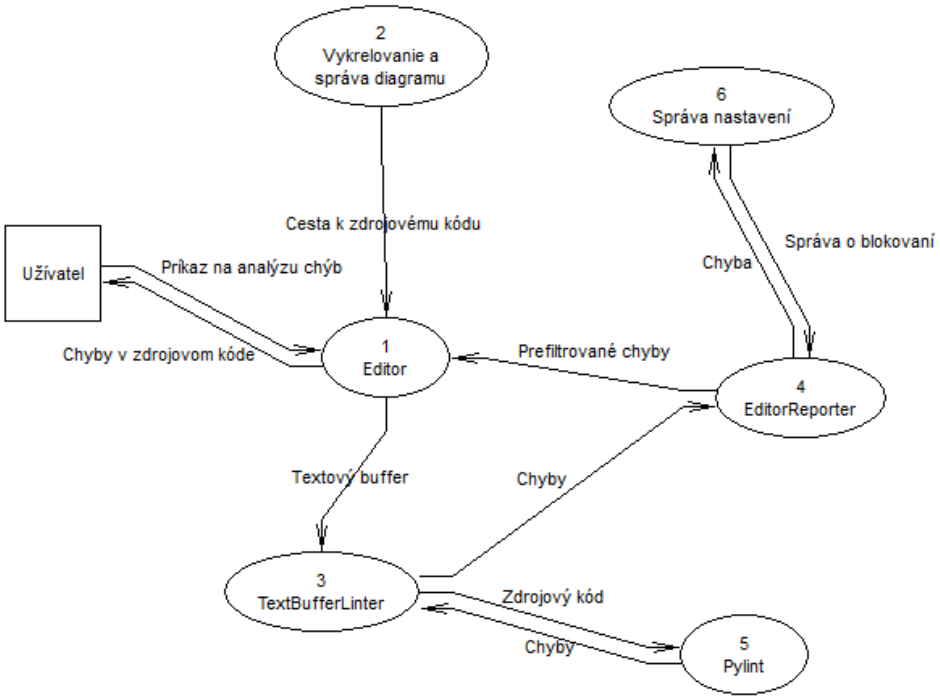
5.4.2 Aplikácia nástroja Pylint

Nástroj Pylint využívame dvomi rôznymi spôsobmi. Potrebujeme jednak preskenovať celý projekt, aby sme chyby pre jednotlivé triedy vykreslili na plátno a zároveň potrebujeme preskenovať práve otvorený súbor. Pre každý tento účel vytvoríme triedy implementujúce funkcionalitu triedy PyLinter pomocou kompozície, ktoré budú bežať v samostnatnom vlákne.

Pri kontrole samostatného súboru funkcionalitu analýzy chýb rieši každý editor samostatne. V nasledujúcom odseku popíšeme spôsob akým túto funkcionalitu implementuje grafický editor využívajúci komponentu GtkSourceView. Ten vytvorí inštanciu triedy TextBufferLinter, ktorú sme naimplementovali tak aby jednotlivým triedam kontrolujúcim kód predala obsah textového bufferu, ktorý obdržala pri inicializácií. Nakoľko každá

inštancia triedy PyLinter potrebuje triedu zodpovednú za výstup výsledkov naimplementovali sme triedu EditorReporter.

Trieda EditorReporter po obdržaní chybovej správy skontroluje, či nieje užívateľom blokovaná. Ak blokovaná nieje vykreslí chybu v zdrojovom kóde.



Pri kontrole celého projektu sa vytvorí inštancia triedy ProjectLinter, ktorej sme nastavili triedu CanvasReporter ako triedu zodpovednú za výstup. Trieda CanvasReporter pri obdržaní správy skontroluje či nieje blokovaná a následne ju nastaví jednotlivým položkám na plátne. Tie obdrží jeho kontextu.

5.4.3 Aplikácia nástroja Gaphas

Nástroj Gaphas použijeme na vykresľovanie diagramu tried na plátno. Jeho jedinou nevýhodou je prítomnosť len základných elementov ako úsečka, obdĺžnik alebo text. Vďaka výbornému návrhu knižnice Gaphas ich ale môžeme využiť na zostavenie vlastných elementov. Z triedy gaphas.item.Element odvodíme triedu Box, ktorá bude reprezentovať triedu. Bude taktiež ucho-

vávať informácie o triede ktorú reprezentuje ako je cesta k súboru, názov, číslo riadku jej výskytu ale aj informácie o chybách, ktoré detekujeme.

Vzťahy medzi triedami vykreslíme za pomoci triedy AssociationLine, ktorá spája dve triedy Box. Podľa vzťahu medzi jednotlivými triedami vykreslí vzor typický pre špecializáciu alebo kompozíciu.

Jednotlivé komponenty reprezentujúce triedy môžeme po plátne presúvať. Z tohto dôvodu sme boli nútení vytvoriť obmedzenie, ktoré bude udržiavať asociačnú čiaru medzi danými dvomi triedami. Triedu sme nazvali HandlesConstraint. Pri pohybe komponenty si parametricky vyjadrí úsečku medzi stredmi spájaných objektov a nájde prieniky medzi ich hranami. Na dané prieniky umiestni konce spájanej úsečky.

Nakoľko chceme vytvoriť rozhranie čo najviac interaktívne potrebujeme nástroj, ktorý po dvojkliku na položku plátna zobrazí jej zdrojový kód v nastavenom editore.

Knižnica Gaphas ponúka výborné rozhranie pre tvorbu vlastného nástroja. Nástroj OpenEditorTool odvodíme od triedy gaphas.tool.Tool a v metóde on_double_click(self, event) pristúpime k vykreslenej položke na plátne nad ktorou je umiestnená myš. Z jej atribútov zistíme cestu k zdrojovému kódu a kód zobrazíme.

5.4.4 Aplikácia ostatných nástrojov

Pre potreby aplikácie potrebujeme využiť nasledujúce nástroje:

- ConfigParser
- cPickle
- OptionParser

Pre uloženie nastavení použijeme triedu ConfigParser, ktorá umožnuje bezproblémovú prácu s nastaveniami. Jej chovanie sme pre aplikáciu prisôsobili vytvorením manažérov nastavení, ktoré implementujeme pomocou singleton návrhového vzoru. Takýmto spôsobom implementujeme nastavenia blokovania Pylint chýb, nastavenie použitého editora alebo uchovanie naposledy otvoreného projektu.

Vlastnosťou nášho nástroja je aj ignorovanie konkrétnych chýb na jednotlivých riadkoch, ktoré ale nechceme ignorovať globálne. Informácie o nich teda uložíme do datového kontajneru, ktorý pomocou nástroja cPickle serializujeme. Pri opätovnom spustení aplikácie prevedieme deserializáciu deserializovaný objekt neskôr použijeme na prefiltrovanie chybových správ.

OptionParser použijeme na získanie informácií, ktoré užívateľ zadá na príkazovom riadku. Trieda OptionParser poskytuje veľmi jednoduché rozhranie na prácu s parametrami príkazového riadka ako je napríklad automatická generácia nápovedy. Pomocou príkazového riadku bude môcť užívateľ zadať napríklad cestu k projektu, ktorý sa má otvoriť alebo typ použitého editoru.

6 Záver

Našou úlohou bolo analyzovať nástroje na analýzu a vizualizáciu Python kódu a vytvoriť nástroj, ktorý poskytuje funkcionalitu, ktorú doteraz vytvorené nástroje neposkytujú.

Po predstavení základných vlastností jazyka Python sme preskúmali vlastnosti jeho introspekcie, ktoré nám vytvorili prehľad o možnosti preskúmať vlastnosti jednotlivých objektov.

Neskôr sme si predstavili nástroje na analýzu kódu PEP8, PyChecker a Pylint a zhodnotili sme, že najvhodnejší nástroj pre backend našej aplikácie je program Pylint, nakoľko je ľahko prispôsobiteľný a okrem detekcie chýb poskytuje funcionalitu vygenerovania grafu. Navyše je široko používaný či už ako samostatná utilita alebo implementáciou v rôzných IDE moduloch.

Analyzovali sme taktiež existujúce nástroje na vizualizáciu kódu a zhodnotili sme ich či už po stránke funkcionality, kompaktnosti alebo iných kritérií.

Nakoniec sme preskúmali a vybrali nástroje, ktoré nám pomohli k ľahšej implementácií projektu a projekt sme navrhli a implementovali.

Implementovaný program nám umožnuje zobraziť ľubovoľný Python projekt v podobe UML diagramu. Výsledný UML diagram sa výzorovo podobá na diagram vygenerovaný programom Graphviz no poskytuje nám okrem iného interaktívnosť a možnosť zobraziť chyby v rámci hierarchie modulov a tried. Pomocou nastavení máme možnosť ignorovať či už celé skupiny chýb, konkrétne typy chýb či ignorovať chyby len na konkrétnych riadkoch. Zdrojové kódy môžeme upravovať v jednoduchých editoroch vizuálneho alebo konzolového charakteru. Aj vďaka zvoleniu vhodných knižníc program veľmi dobre škáluje a je spustený a inicializovaný za krátky čas.

Medzi nevýhody programu môžeme zaradiť fakt, že slúži na analýzu len jedného programovacieho jazyka alebo fakt, že niektorá funkcionalita ako je napríklad filtrácia chýb by mohla byť implementovaná na nižšej úrovni aby bola prístupná aj spustením z príkazového riadku bez spustenia grafického rozhrania.

V budúcnosti sa program bude môcť spustiť na iných operačných systémoch medri ktoré patrí napríklad aj Microsoft Windows. Ďalej bude možné viac prispôsobiť program pre jednoduchšie editovanie zdrojových kódov akými sú napríklad klávesové skratky a iné vlastnosti pokročilých editorov.

7 Príloha

Pylint obsahuje nasledujúce triedy typu checker

- ImportsChecker
- NewStyleConflictChecker
- ClassChecker
- BaseChecker
- BaseRawChecker
- BasicErrorChecker
- NameChecker
- DocStringChecker
- PassChecker
- MisdesignChecker
- ExceptionsChecker
- TypeChecker
- LoggingChecker
- StringFormatChecker
- FormatChecker
- VariablesChecker
- RawMetricsChecker
- SimilarChecker