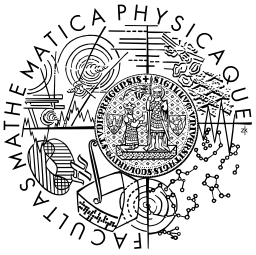


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Dominik Hrdý

PerfEval: Spojení unit testů s vyhodnocováním výkonu

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: prof. Ing. Petr Tůma, Dr.
Studijní program: Informatika
Studijní obor: Systémové programování

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: PerfEval: Spojení unit testů s vyhodnocováním výkonu

Autor: Dominik Hrdý

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: prof. Ing. Petr Tůma, Dr., Katedra distribuovaných a spolehlivých systémů

Abstrakt: Abstrakt.

Klíčová slova: testování výkonnost

Title: PerfEval: Marrying unit testing with performance evaluation

Author: Dominik Hrdý

Department: Department of Distributed and Dependable Systems

Supervisor: prof. Ing. Petr Tůma, Dr., Department of Distributed and Dependable Systems

Abstract: Abstract.

Keywords: testing performance

Obsah

Úvod

Po více než dvacet let pomáhají unit testy udržovat kvalitu kódu v průběhu vývoje softwaru. Za tuto dobu bylo vyvinuto spoustu knihoven pro implementaci a spouštění unit testů. Verzovací nástroje jako GitLab, nebo GitHub umožňují v rámci vývoje software verzovat, spouštět unit testy a reagovat na jejich případná selhání.

Pro udržování korektnosti kódu využíváme pokrytí unit testy. Udržování korektnosti je dnes již naprosto běžnou součástí vývoje softwaru. Psaní výkonnostních testů ovšem tak běžné není. Práce s frameworky pro měření výkonu totiž není tak jednoduchá jako používání knihoven pro psaní unit testů, a proto jejich užití není tak časté.

Frameworky pro vyhodnocování výkonu softwaru jsou podobné knihovnám pro psaní unit testů. Frameworky jako JMH nebo BenchmarkDotNet pomáhají implementovat výkonnostní testy. Výkonnostní testy se implementují obdobně jako u unit testů obvykle pomocí anotací. Vyhodnocování naměřených výsledků dnes obvykle zahrnuje i ruční zhodnocení naměřených výsledků. Vyhodnocování výkonu totiž vyžaduje porovnat naměřený výkon s nějakou další referenční hodnotou. Měřící frameworky tyto referenční hodnoty nemají a ani je nikde získat nemohou, protože měří pouze jednu verzi softwaru. Nemohou tedy dělat zmíněné celkové vyhodnocení.

Z porovnání vyhodnocování unit testů a výkonnostních testů, není vyhodnocování výkonnostních testů tak jednoduché. Unit testy testují korektnost. Korektnost se prokáže tak, že na každý zadaný vstup program vrátí očekávaný výstup. Při vyhodnocování výkonu se změří nějaká data o běhu programu. Tato data ale sama o sobě nic neříkají a je nutné je zkoumat v kontextu.

Výsledkem této práce je nástroj nazvaný PerfEval. PerfEval je konzolová aplikace napsaná v programovacím jazyce Java. Perfeval umí pomocí argumentů na příkazové řádce vyhodnocovat výsledky výkonnostních testů.

PerfEval je schopen automatického hlášení výsledků výkonnostních testů. Umí porovnávat výsledky měření výkonu dvou verzí softwaru mezi sebou. PerfEval je také vhodný pro skriptování, protože o výsledcích informuje nejen výpisem na standardní výstup, ale také exit kódem. Nástroj podporuje zpracování výstupů frameworků BenchmarkDotNet a JMH ve formátu JSON, ale je rozšiřitelný i pro zpracování jiných frameworků, nebo formátů.

1. Kontext

1.1 Testování softwaru při vývoji

Při vývoji softwaru je vhodné vyvíjený software neustále testovat. Software lze testovat více způsoby, ale cílem testů je vždy otestovat některý z důležitých kvalitativních atributů jako je například korektnost, nebo výkon.

Pro testování korektnosti se obvykle používají unit testy. Jedná se většinou o krátké testovací funkce, které kontrolují jestli se testovaný kód chová požadovaným způsobem. Zjišťují tedy jestli kód na zadaný vstup vrátí očekávaný výstup. Ke kódu který není aktuálně vyvíjen, se obvykle nemění ani unit testy, a proto umožňují udržovat neustálou korektnost již hotového kódu. Zda-li je kód korektní se pomocí unit testů zjistí velmi jednoduše. Kód je korektní pokud vrátil očekávaný výstup.

Testování výkonu obvykle probíhá tak, že se použije nějaký vhodný měřící framework. Obvyklé frameworky pro měření výkonu mají vlastní pravidla, jak se mají oanotovat metody, které se mají měřit. Tyto frameworky umožňují měřit výkon softwaru podle různých metrik. Mezi tyto metriky se řadí čas, propustnost a například spotřeba paměti. Výsledky měření frameworky umí obvykle zaznamenat jak do strojově čitelného formátu, tak do formátu čitelného pro člověka. Výsledek měření je ale pouze sada čísel, kde jsou ke jménům testovaných metod přiřazeny naměřené hodnoty.

TODO: doplnit obrázek nějakého výstupu například BenchmarkDotNet, nebo JMH? možná by se hodilo nějak vhodně zasazený přímo textový výstup

Výsledky testování výkonu se tedy musejí vyhodnocovat tak, že se podrobně prozkoumá výsledná sada čísel. Oproti testování korektnosti, kdy test projde, nebo neprojde je testování výkonu výrazně složitější. Pohledem na samostatnou sadu dat se nedá určit zda-li je software dostatečně rychlý. Aby bylo možné ze sady určit něco vypovídajícího mohl by být stanoven limit výkonnosti. Například by se stanovilo, že metoda se nebude konat déle než 5 s. Tento přístup nemusí být vypovídající při dlouhodobém vývoji a při časech hluboko pod limitem. Proto je vhodné aby se datové sady, které testovací framework produkuje, testovaly navzájem mezi sebou. Porovnáním sad je totiž možné zjistit, jestli nedošlo k významným změnám výkonu při vývoji od předchozí verze. Frameworky samotné však tuto možnost obvykle nemají.

1.2 Průběžná integrace

Při vývoji softwaru se čato používají nástroje pro automatizování některých činností. Obvykle se automatizují činnosti jako jsou správa verzí, spouštění testů a jejich vyhodnocování.

Pro správu verzí při vývoji softwaru se obvykle používá technologie git. Jedná se o sytém pro správu verzí. Git je vhodný i pro práci ve velkých týmech. Umožňuje totiž členění projektu do větví. Každá větev je vhodná pro vývoj samostné části aplikace a následným spojováním větví pak dochází k propojení funkčností vyvinutých v jednotlivých větvích. Technologie si formou pamatování si změn udržuje přehled o průběžných verzích a umožňuje uživatelům (programátorům)

vrátit se vývoji zpět. Technologii je možné ovládat jednoduchými příkazy z příkazové řádky. Je tedy vhodná pro psaní krátkých skriptů.

Nástroj git umí jednoduše zpostředkovat průběžnou integraci (continuous integration). Průběžná integrace umožňuje dělat uživateli automatizované kroky při nahrávání nových verzí do větve nebo při spojování větví. Při průběžné integraci tedy jde o spouštění skriptu při některé ze zmíněných událostí. V rámci průběžné integrace je možné testovat software pomocí unit testů i benchmarků pro měření výkonu. Dále je možné použít jakékoli jiné příkazy příkazové řádky. Do průběžné integrace je tedy možné jednoduše zapojit téměř jakoukoli konzolovou aplikaci.

1.3 Scénář

Mějme programátora, který se rozhodne vyvíjet nový software. Projekt se mu začne rozrůstat, a tak aby mohl udržovat větší projekt napíše si unit testy. Pomocí unit testů si udržuje průběžnou korektnost již hotového kódu. Dále pojme podezření, že se jeho stávající kód začíná zpomalovat (klesá výkon). Proto se rozhodne použít framework pro testování výkonu a spustí testy pro měření výkonu svého kódu.

Programátor změří výkon na verzi před tím než pojmul podezření o zpomalování. Následně změří výkon aktuální verze. Výsledkem těchto měření jsou dvě tabulky s daty. V každé z nich je jméno testovací metody k níž je přiřazen výsledek měření. Protože ale benchmarkovací framework funguje jako stopky, tak není schopný porovnat dvě tabulky mezi sebou. Analogie ke stopkám je, že jsou schopny změřit čas prvního a druhého kola zvlášť, ale nejsou schopny je porovnat.

Programátor se tedy musí sám podívat, jestli výkon jeho softwaru klesá. Programátor by tedy potřeboval nástroj, který by uměl vzít výstupy měření z různých verzí a porovnal je mezi sebou. Zároveň by programátor jistě chtěl, aby byl na zhoršení výkonu upozorněn podobným způsobem jako na selhání unit testů tj. selháním průběžné integrace.

TODO: dodat příklad skriptu průběžné integrace

2. Problém vyhodnocování výkonu

2.1 Automatické vyhodnocování

V minulé kapitole byl zmíněn problém programátora s testováním výkonu. Programátor má unit testy, které testují korektnost jeho softwaru. Umí je efektivně spouštět s každou změnou pomocí průběžné integrace. Chtěl by, aby mohl podobně efektivně testovat i výkon svého softwaru.

S testováním výkonu je ale problém. Když se spustí měření výkonu, tak výsledkem je pouhá datová sada. Tato datová sada nevypovídá nic o změně průběžného výkonu, která je zajímavá. Právě podle změny ve výkonu softwaru je možné zjistit, jestli je nutné kód optimalizovat, protože dochází k významným zhoršením.

Při samotném měření výkonu je ale možné, že se některé hodnoty výrazně odchýlí od ostatních. Může tedy být nutné měření opakovat a z více naměřených hodnot dohromady usuzovat, jak se výkon v průběhu vývoje mění. K analýze takto získaných hodnot se ale hodí využít statistických metod, které si i se zmíněnými odchylkami umí poradit.

Vyvinutý systém PerfEval řeší programátorův problém. Jedná se o nástroj, který může zakomponovat do své průběžné integrace tak, aby byl pokles výkonu hlášen. Nástroj při spuštění průběžné integrace porovná dvě poslední verze a případně oznámí zhoršení výkonu. Na základě tohoto hlášení může selhat celá průběžná integrace, což je obdobné chování, jako se očekává při selhání unit testů.

2.2 Výsledky měření výkonu

Z počátku je dobré se zamyslet nad tím, jak můžou výsledky měření výkonu softwaru vypadat. První domněnky o tom, že stačí měřit pouze čas se ukázaly jako mylné. Testovací frameworky totiž umožňují mimo měření času také měření frekvence nebo například spotřeby paměti. Předpokládat se tedy dá jen to, že pokud vezmu dva výsledky měření výkonu ze dvou různých verzí, tak budou reprezentovány stejnou fyzikální veličinou a v lepším případě budou mít i stejnou fyzikální jednotku.

Protože testovací frameworky nepoužívají žádné identifikátory testů, tak je nejpřímější řešení jejich rozpoznávání používat jména testovacích metod jako identifikátor. Tato jména poskytují ve výsledcích měření jak framework BenchmarkDotNet, tak framework JMH. Z dokumentace frameworku Criterion pro měření výkonu v jazyce R se název metody ve výsledcích nachází také. Z toho lze usoudit, že použití jména metody jako identifikátoru může být dostatečně obecné. (DOKUMENTACE - CRITERION)

2.2.1 Výstup měření BenchmarkDotNet

BenchmarkDotNet je framework určený k měření výkonu programů na platformě .NET. V důsledku toho, že měří programy na platformě .NET je schopen

měřit výkon programů napsaných v programovacích jazycích C#, F# a Visual Basic (DOKUMENTACE - BenchmarkDotNet README.md).

Při měření výkonu pomocí BenchmarkDotNet se měřené hodnoty vypisují na standardní výstup včetně konečného shrnutí. Mimo standardní výstup se ještě výsledky měření ukládají do strojově zpracovatelných formátu jako je například JSON, nebo CSV.

(DODAT PŘÍLOHU)

Měření výkonu programu v jazyce C# pomocí frameworku BenchmarkDotNet má za výstup již statisticky zpracované hodnoty. Aby bylo možné sledovat jednotlivé naměřené hodnoty je nutné zvolit již při psaní testů správný exportér, který tuto funkci podporuje. Pro možnost sledování konkrétních naměřených hodnot u exportéru nazvaného JsonExporter je nutné nastavit možnost excludeMeasurements na false, aby se ve výstupním souboru hodnoty objevily. Jedna naměřená hodnota se nachází u výsledku měření jako položka Nanoseconds. Z názvu položky lze usoudit, že v základním nastavení BenchmarkDotNet se měří čas.

Celkově tedy v souboru s výsledky nalezneme podrobnosti o prostředí jako je operační systém, verze platformy .NET, jméno, typ a parametry procesoru. Dále se v souboru s výsledky nachází výsledky jednotlivých provedených měření. Výsledek měření u sebe má informaci o jméně testovací metody, zpracované statistické údaje a naměřené hodnoty z různých módů a iterací měření.

Je nutné vybírat jen některé hodnoty, protože C# je kompilovaný jazyk metodou JIT. Je tedy nutné dávat pozor jaká data pro porovnávání výsledků budou použita. Určitě není správně dávat k sobě pro porovnání hodnoty, které byly naměřeny před optimalizacemi překladačem a po nich. Může totiž dojít ke zkreslení výsledků.

TODO: přidat jako přílohu výstupy BenchmarkDotNet odkazovat se do nich a do dokumentace BDN

2.2.2 Výstup měření JMH

JMH je framework pro měření výkonu, který umožňuje pomocí anotací definovat výkonostní testy pro programy v jazyce Java. Z průzkumu (ZDROJ-prezentace) vyplývá, že se jedná o nejpoužívanější framework pro měření výkonu pro projekty vyvíjené v jazyce Java.

JMH obdobně jako BenchmarkDotNet poskytuje výsledek měření jako tabulku na standardní výstup. Dále poskytuje výstup také v podobě strojově zpracovatelných formátů jako jsou například XML, nebo JSON. O výstup v této podobě je nutné zažádat pomocí argumentů na příkazové řádce při spouštění měření (DOKUMENTACE - JMH).

(DODAT PŘÍLOHU)

Ve výstupním souboru měření pomocí JMH lze nalézt informace o stroji na kterém probíhalo měření. Jedná se především o název stroje a verzi operačního systému. Dále zde lze nalézt verzi Javy, ve které probíhalo měření. V souboru je možné vidět také ostatní parametry měření, jako je zahřívací doba a počet zahřívacích iterací. Zahřívací iterace jsou důležité opět kvůli tomu, že Java je překládaná technologií JIT.

Jednotlivé naměřené hodnoty jsou ve výstupním souboru dostupné i ve výchozím nastavení JMH. Naměřené hodnoty jsou reprezentovány pomocí metriky.

V rámci metriky jsou uloženy informace o fyzikální jednotce, která se měřila, a sady naměřených hodnot. Pro každý běh (fork), přičemž běhů může být v jednom spuštění JMH více, se objeví jedna sada měřených hodnot u metriky.

2.3 Použití statistických metod pro analýzu dat

Pro vyhodnování výkonu je využito metod testování hypotéz. Ve statistickém testování hypotéz se snažíme zamítnout nulovou hypotézu. V případě zamítnutí nulové hypotézy se předpokládá, že platí alternativní hypotéza.

Při testování hypotéz rozeznáváme chyby I. a II. druhu. Chyba I. druhu znamená, že jsme nulovou hypotézu zamítli, i když platí. Chyba II. druhu znamená, že jsme ji nezamítli, ale ona neplatí.

V našem případě porovnávání výkonu bude nulová hypotéza tvrzení, že výkon obou verzí je stejný. Jako alternativní hypotézu budeme uvažovat, že výkony obou verzí jsou různé. Naměřené hodnoty jedné testovací metody každé z verzí budeme považovat za diskrétní náhodnou veličinu. Pokud hodnoty těchto veličin mají stejné rozdělení, tak budeme tvrdit, že i výkon obou verzí je stejný.

Chyba I. druhu tedy v našem případě znamená, že jsme prohlásili, že výkony dvou verzí jsou různé, ačkoli jsou stejné. Pravděpodobnost chyby I. druhu je obvyklý parametr statistického testu. Pravděpodobnost chyby I. druhu bude dále značen jako parametr α. Parametr α je součástí konfigurace systému PerfEval.

2.3.1 Dvouvýběrový T-test

Dvouvýběrový T-test se používá při neznalosti rozptylu zkoumaných náhodných veličin. Protože není možné usuzovat rozptyl naměřených hodnot, tak nám tato vlastnost vyhovuje. Dvouvýběrový T-test zkoumá jestli je rozdíl středních hodnot δ. Pokud mají být rozdělení náhodných veličin stejná, rozdíl jejich středních hodnot by měl být nula ($\delta = 0$).

T-test vyžaduje od náhodných veličin tři vlastnosti. Měly by být nezávislé, normální a měly by mít stejný rozptyl. Nezávislost veličin není možné při měření zajistit, proto bude případná závislost zanedbána. Stejný rozptyl obou veličičn také není možné zaručit. Lze ale předpokládat, že měření veličin při stejných podmínkách, způsobí podobný rozptyl. Podle shrnutí o dvouvýběrových testech (ZDROJ - dvouvýběrové testy) způsobí zanedbání shodnosti rozptylů pouze malou chybu.

Posledním z požadavků dvouvýběrového T-testu je normalita náhodných veličin. Toho docílíme pomocí centrální limitní věty (CLV). Výsledky měření se skládají z běhů. Každý běh obsahuje naměřené vzorky náhodné veličiny, která reprezentuje naměřenou hodnotu výkonu. Pokud budeme v rámci T-testu namísto naměřených hodnot uvažovat průměry jednotlivých běhů, tak se podle CLV bude rozdělení těchto průměrů blížit normálnímu rozdělení.

Dvouvýběrový T-test je dnes běžnou součástí standardních matematických knihoven. Použití T-testu v programu je tedy velmi jednoduché, protože není potřeba zamýšlet se nad korektností psaného kódu. V případě systému PerfEval se T-test provádí tak, že se z naměřených hodnot vyrobí Studentovo T-rozdělení. Z tohoto rozdělení se zjistí intervalový odhad střední hodnoty.

Nakonec se již jen zkoumá zda-li tento interval obsahuje nulu ($\delta = 0$). Pokud interval nulu neobsahuje, pak lze tvrdit, že s pravděpodobností $1 - \alpha$ nulová hypotéza neplatí.

TODO: odkaz do skript z PaSt1, doplnit nějakou myšlenku?

2.3.2 Hierarchický bootstrap

Bootstrap je statistická metoda využívající tzv. resamplování. Stejně jako u dvouvýběrového T-testu se předpokládá nezávislost náhodných veličin. Podmínka nezávislosti bude v systému zanedbána, protože ji není možné zaručit. Bootstrap se jako metoda používá v případě, kdy se o náhodných věličinách nedá dopředu téměř nic usuzovat. Toto odpovídá případu měření výkonu, kdy neznáme ani rozptyl ani střední hodnotu.

Mějme naměřený statistický soubor dat o velikosti n. Z tohoto statistického souboru vybereme n-krát náhodný prvek. Z těchto náhodně vybraných prvků spočítáme průměr. Spočtený průměr budeme považovat za nový vzorek nově počítaného statistického souboru. Namísto malého statistického souboru se tedy vytvoří větší statistický soubor. Z nového statistického souboru se určí interval spolehlivosti tak, že se vezme $\frac{\alpha}{2}$-tý a $\frac{1-\alpha}{2}$-tý percentil.

Zkoumanou náhodnou veličinou je rozdíl dvou náhodných veličin. Tyto dvě náhodné veličiny jsou dány měřením výkonnosti dvou verzí softwaru. Nulová hypotéza, která je vyvracena, tvrdí, že obě veličiny mají stejné rozdělení. Pokud tedy interval spolehlivosti neobsahuje nulu, tak můžeme nulovou hypotézu vyvrátit, protože s pravděpodobností $1 - \alpha$ neplatí.

Naměřené vzorky však nejsou prostý statistický soubor. Jedná se o hierarchický soubor dat. Každé jedno měření se skládá z jednoho, nebo více běhů. Každý běh se skládá z jednoho, nebo více naměřených údajů. Vytváření bootstrapového statistického souboru tedy vypadá trochu odlišně. Prvky tohoto souboru budou spočteny z náhodně vybraných běhů. Hodnota prvku náhodně vybraného běhu bude průměr bootstrapového souboru spočteného z měření v tomto běhu. Bootstrapový soubor z hodnot běhu se již spočte výše zmíněným způsobem. Vybere se n náhodných hodnot z naměřených hodnot v běhu.

2.3.3 Co dělat v případě nevyvrácení hypotézy?

V případě nevyvrácení nulové hypotézy nám statistické testy nedávají žádnou informaci. Nicméně je stále nutné se rozhodnout zda-li nulová hypotéza platí. Je nutné se ale stále vyvarovat chybě II. druhu. Proto v tomto případě budeme považovat nulovou hypotézu za platnou, pokud bude interval spolehlivosti dostatečně úzký. Pokud má tedy interval spolehlivosti dolní mez 1 a horní mez 3, pak je jeho délka 2. Odhadovaný průměr by byl také 2. Relativní úzkost intervalu by tedy byl poměr průměru a délky, tedy 1.

V případě většího množství vzorků je možné zužovat interval spolehlivosti. Vztah mezi úzkostí a počtem vzorků odpovídá $O(\frac{1}{\sqrt{n}})$, kde n je počet vzorků. Z daného množství vzorků je tedy možné analyticky určit, kolik vzorků je ještě zapotřebí změřit.

V případě, že je interval dostatečně úzký prohlásíme, že nulová hypotéza platí. V případě, že interval není dostatečně úzký, prohlásíme, že vzorků není dost.

3. Systém PerfEval

3.1 Popis systému

Systém PerfEval je konzolová aplikace, jejíž činnost je ovládaná příkazy a parametry na příkazové řádce. Příkazy umožňují inicializovat systém, přidávat nové výsledky měření a vyhodnocovat mezi sebou výsledky měření dvou verzí.

3.1.1 Průběh vyhodnocování

Hlavní funkcionalitu pro vyhodnocování poskytuje třída EvaluateCLICommand. Celý úkol vyhodnocování výkonu byl rozdělen do tří oddělených kroků.

Jako první dojde k naparsování výsledků měření dvou porovnávaných verzí. Parsování probíhá pomocí implementace rozhraní MeasurementParser, který umí naparsovat potřebný formát výsledků měření, které se budou porovnávat. Výsledkem parsování vzniknou dvě instance třídy Samples, která reprezentuje vzorky výsledků měření.

Následně se tyto dvě instance porovnají pomocí metody compareSets na třídě PerformanceEvaluator. Instance třídy PerformanceEvaluator k porovnání dvou instancí Samples využívá implementace rozhraní StatisticTest. Konkrétně tato implementace rozhoduje o tom, jaký statistický test se při porovnávání vzorků použije.

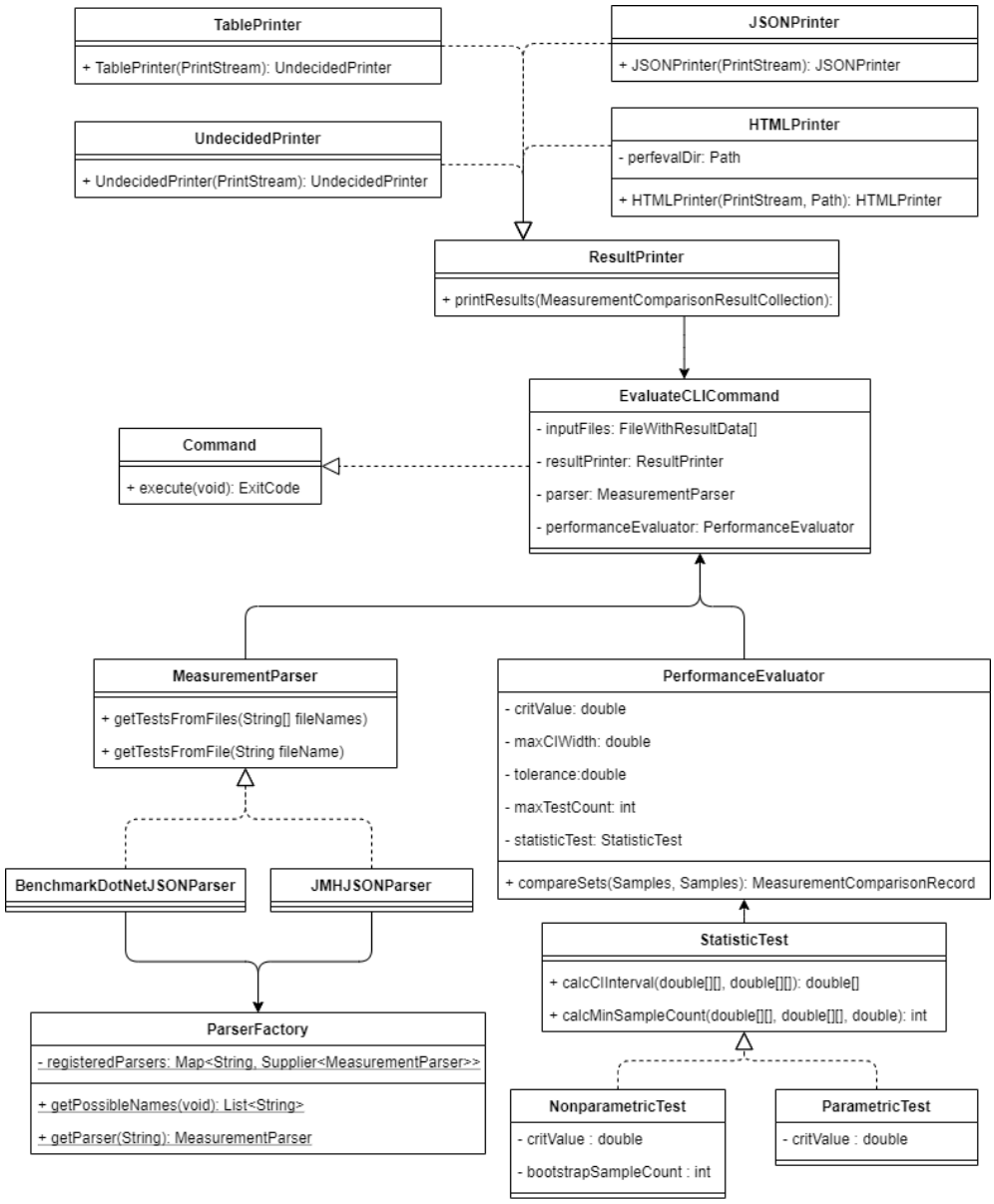
Statistický test, který implememtuje rozhraní StatisticTest, musí mít metody calcCIInterval a calcMinSampleCount. Metoda calcCIInterval vrací intervalový odhad rozdílu náhodných veličin, které jsou reprezentovány konkrétními naměřenými vzorky.

Pokud tento interval neobsahuje nulu, pak lze s požadovanou přesností a pravděpodobností usuzovat, že rozdělení náhodných veličin jsou různá. Pokud jsou rozdělení různá, pak se jako kladný výsledek testu připouští zlepšení výkonu, nebo pokud je nový průměr v dané toleranci.

Pokud interval obsahuje nulu, tak je vyhodnocení výsledku testu výkonu složitější. Pokud je interval spolehlivosti dostatečně úzký, tak považujeme rozdělení náhodných veličin za stejná a test výkonu dopadl kladně. Pokud interval spolehlivosti není dostatečně úzký, tak test výkonu neprošel. Následně se pomocí metody calcMinSampleCount určí kolik měření by bylo potřeba, aby byl interval spolehlivosti dostatečně úzký. Spočtený počet vzorků s informací o tom, zda-li je, nebo není možné je změřit, bude přidán k výsledkům porovnávání.

Posledním krokem při vyhodnocování je zavolání metody printResults na implementaci rozhraní ResultPrinter. Tato metoda vypíše instanci MeasurementComparisonResultCollection, která reprezentuje všechny výsledky jednotlivých testů výkonu. Po vypsání výsledků požadovaným způsobem dle dodané implementace ResultPrinter se nastaví správný exit kód. V případě, že všechny testy dopadly kladně, pak bude exit kód 0. V případě, že alespoň v jednom případě došlo ke zhoršení, pak bude exit kód 1. V případě, že dojde k nějaké výjimce, tak exit kód bude větší než 100. Touto výjimkou může být například neexistence některého ze souborů s naměřenými výsledky v důsledku jeho smazání, nebo přesunutí.

Na následujícím obrázku je k vidění architektura části systému právě kolem třídy EvaluateCLICommand, která zajišťuje průběh vyhodnocování testů výkonu.



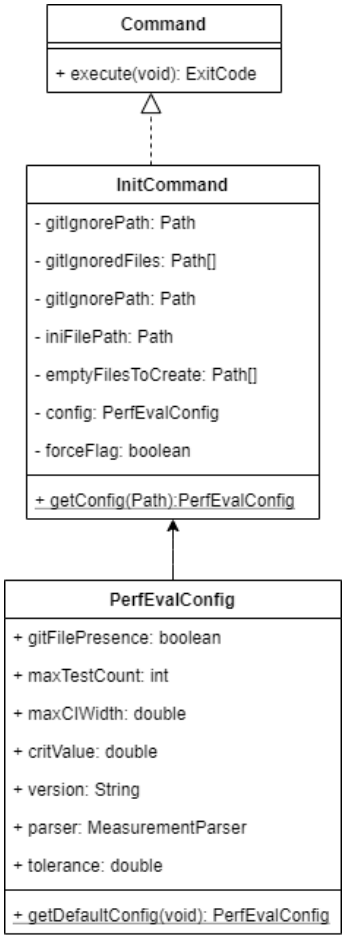
Obrázek 3.1: Objektový návrh časti PerfEvalu pro porovnávání výsledků měření

Hlavní myšlenka návrhu je tok dat programem. Po zpracování příkazové řádky dostane instance EvaluateCLICommand informace o soubourech se kterými bude pracovat. Při konstrukci dostane správné implementace ResultPrinter, MeasurementParser a instanci třídy PerformanceEvaluator se správnou instancí StatisticTest. Data plynou programem tak, že se naparsují pomocí implementace třídy MeasurementParser. Naparsované výsledky se pomocí statistického testu vyhodnotí mezi sebou. Výsledky porovnání se vypíšou pomocí implementace rozhraní ResultPrinter.

3.1.2 Inicializace systému PerfEval

Před použitím systému PerfEval k vyhodnocování je nutné jej inicializovat. Inicializace umí rozpoznat zda-li se nachází v kořenovém adresáři Git repozitáře. Rozpoznávání probíhá tak, že se hledá soubor s názvem .git.

Inicializace probíhá tak, že se zjistí přítomnost .git souboru. Z příkazové řádky se zjistí jaký parser (implementace třídy MeasurementParser) pro naměřené hodnoty se bude používat. Vyrobí výchozí konfigurace. Do výchozí konfigurace se dodají informace o git repozitáři a o parseru. Nakonec se vyrobí složka se v adresáři, kde uživatel program spustil vyrobí složka .performance a v ní soubory .gitignore a config.ini. Soubor config.ini obsahuje konfiguraci systému PerfEval pro jeden projekt jehož výkon mezi verzemi se bude porovnávat.



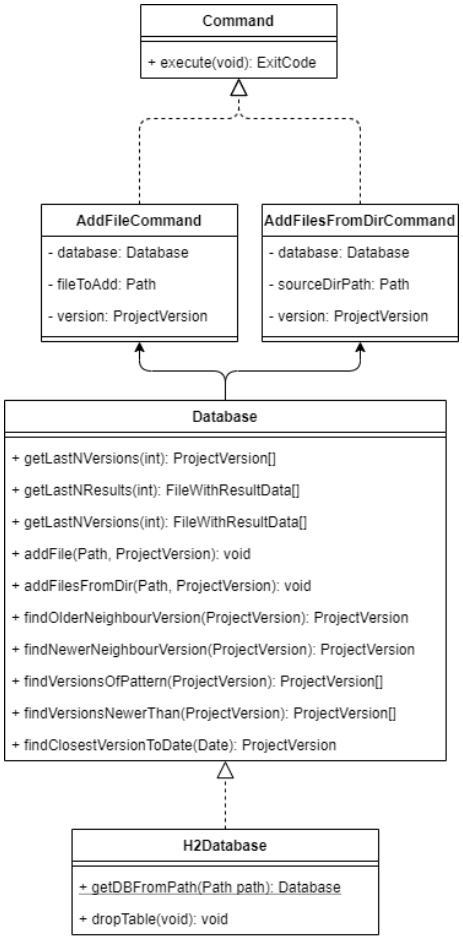
Obrázek 3.2: Objektový návrh časti PerfEvalu pro inicializaci

Z obrázku je patrné, že třída InitCommand úzce spolupracuje s třídou PerfEvalConfig. Třída PerfEvalConfig reprezentuje obsah souboru config.ini. Provedení metody execute na třídě InitCommand provede inicializaci systému PerfEval výše zmíněným způsobem.

3.1.3 Přidávání nových výsledků testů

Uživatel musí každý výsledek měření výkonu zaznamenat do systému PerfEval. Pokud výsledek měření zaznamenán, nebude s ním PerfEval vůbec pracovat.

Přidávat je možné soubory samostatně, nebo z adresáře. Při přidávání souborů z adresáře budou přidány všechny soubory a to i soubory zanořené ve vnitřních adresářích zadaného adresáře.



Obrázek 3.3: Objektový návrh časti PerfEvalu pro přidávání výsledků měření

Na obrázku je vidět, že za oběma příkazy pro přidávání výsledků měření stojí jedna implementace rozhraní Database. Jediná existující implementace tohoto rozhraní využívá technologie H2 databáze. Jedná se o technologii, která umožňuje vést si databázi lokálně v rámci souboru a pokládat na ní klasické SQL dotazy.

Implementace rozhraní Database pomocí technologie H2 je třída H2Database. V rámci databáze pro jeden projekt, který PerfEval spravuje jsou lokální soubory pro H2 databázi uloženy v adresáři .performance. Databáze pro správu dat o výsledcích měření má jen jednu tabulku. V této tabulce jsou informace o cestě k souboru a informace o verzi.

3.2 Rozbor alternativ v řešení

Při vývoji systému PerfEval došlo k několika rozhodnutím, které výrazně ovlivnily jeho fungování a architekturu. Nejdůležitější rozhodnutí, která byla v průběhu výoje provedena jsou v následujících částech podrobně vysvětleny.

3.2.1 Grafické rozhraní

PerfEval je aplikace používaná z příkazové řádky. Od počátku plánování bylo zamýšlené, že se bude ovládat pomocí příkazové řádky. Je určen pro porovnávání výsledků testování výkonu dvou verzí softwaru. Pomocí exit kódu signalizuje, zda-li nedošlo ke zhoršení výkonu.

PerfEval měl mít ale i grafickou část. V grafické části by si mohl uživatel prohlížet dlouhodobý vývoj výkonu jeho testovaného softwaru. K dispozici by mu v rámci tohoto rozhraní mělo být porovnání více verzí v podobě grafů. Zároveň by si uživatel mohl omezovat zobrazovaná období pro detailnější zkoumání.

Protože základní aplikace bez grafického rozhraní byla složitější než zpočátku zdálo, tak grafické rozhraní aplikace nebylo realizováno.

3.2.2 Spouštění testů systémem PerfEval

V počátku vývoje bylo nutné se rozhodnout jakým způsobem bude systém přijímat a zpracovávat výsledky testů. V úvahu přicházela varianta, že uživatel provede měření výkonu sám, a druhá varianta, že uživatel systému vysvětlí jakým způsobem se testování výkonu spouští. Pokud by byla zvolena varianta, kdy PerfEval spouští testy sám, tak by bylo nutné nalézt dostatečně univerzální způsob spouštění testů.

Aplikace a benchmarky pro měření výkonu mohou být jak konzolové, tak grafické aplikace. Pokud by PerfEval měl měření provádět sám, tak by téměř určitě nebyl schopen pracovat s grafickými aplikacemi, ale byl by schopen spouštět programy s parametry na příkazové řádce.

Dále by bylo nutné vysvětlit jak vypadá výstup spouštěných testů. Když pomineme formát, tak je nutné zjistit, kam program, který provádí měření výsledky ukládá. Benchmarkovací systém BenchmarkDotNet například vypisuje výsledky měření v podobě tabulky na standardní výstup a zároveň ukládá strojově čitelné výsledky do speciálního k tomu určeného adresáře.

Nakonec by to vypadalo tak, že při inicializaci systému uživatel zadá jak spustit test výkonu a kam se uloží výsledek. Tímto způsobem by došlo k tomu, že PerfEval by začal určovat, jak mají vypadat programy jejichž výstupy přijímá. Na druhou stranu použití stávajícího řešení tyto nevýhody nemá. Jediné, co musí uživatel zajistit je, že jeho způsob měření zvládne uložit výsledek do souboru, který PerfEval umí zpracovat.

Stávající řešení je tedy složitější v tom, že uživatel musí testování výkonu spouštět sám. Poté, co spustí testování výkonu a dostane soubor s výsledky je přidá do databáze výsledků systému PerfEval a poté spustí PerfEval s příkazem evaluate pro porovnání výsledků testování výkonu. Kdyby se použila varianta s tím, že si PerfEval spouští testy sám, tak by uživatel testování a porovnání výkonu zvládl jedním spuštěním aplikace.

3.2.3 Použité statistické metody

Pro porovnání dvou výsledků měření se používají statistické metody. Statistické metody se používají ke zjištění, jestli výsledky měření považované za náhodné veličiny mají stejné rozdělení, popřípadě jestli je vzorků dostatečné množ-

ství. V systému PerfEval jsou implementovány statistické metody bootstrap a T-test.

Pomocí bootstrapu dochází k iluzi, že vzorků je mnohem více, než je jich ve skutečnosti změřeno. Protože měření vzorků je samo o sobě dlouhotrvající proces, tak se použití této statistické metody vyplatí. Bootstrap má ale tu nevýhodu, že jeho výpočet může oproti prostému zkoumání vzorků trvat delší dobu.

T-test je tedy možné zvolit namísto bootstrapu kvůli tomu, že je rychlejší. Oproti náhodnému vybírání vzorků z dvoudimenzionální sady dat, kterým vzorky odpovídají, dochází k pouhému dosazení hodnot do vzorce.

Uživatel by se tedy při volbě testu měl rozhodovat na základě toho kolik má změřených vzorků a kolik má času na porovnání výsledků měření. Pokud má uživatel naměřených vzorků málo a větší množství času k vyhodnocení doporučuje se použít bootstrap. Pokud má uživatel větší množství vzorků a málo času je pro něho lepší použít T-test.

3.2.4 Forma ukládání informací o měřeních

Systém, který provnává výsledky měření, by měl mít nějaké informace o tom, k jaké verzi se měření vztahuje, nebo kde se soubor s výsledky nachází. Proto bylo při vývoji nutné se zamyslet nad tím jakými způsoby lze tyto informace získat. Systém PerfEval potřebuje o výsledcích měření vědět, kde jsou uloženy a ke které verzi softwaru bylo měření provedeno.

První zvažovaná varianta byla, že by existovala složka do které by uživatel vkládal výsledky měření do adresáře určeného systémem PerfEval, nebo do adresáře zadaného při inicializaci systému. Po každém měření by tedy uživatel výsledek akorát vložil do správné složky a o nic víc by se nemusel starat.

Toto řešení má několik nevýhod. Omezuje uživatele v tom, jak musí výsledky testů ukládat. Dále by se špatně určovalo ke které verzi bylo měření provedeno, protože toto se ve výsledcích měření běžnými frameworky neudává. Pravděpodobně by proto vznikla hierarchie v tomto adresáři a uživatel by například pomocí pojmenovávání složek určoval ke které verzi se měření vztahuje.

Celkově tedy tento způsob poskytuje jednoduché zjištění, kde se výsledek testu nachází. Je to pevně dané. Není ale možné jednoduše zjistit verzi, protože je nutné projít adresářovou strukturu, což může být při velkém množství dat pomalé.

Druhá zvažovaná varianta byla taková, že by PerfEval znal celý obsah souborů s výsledky testování. Konkrétně by v nějakém svém adresáři měl uložený celý výstup měření ve svém vlastním formátu, dooplněné o vlastní poteřbná metadata. Při zaznamenání nových výsledků by tedy došlo k načtení celého souboru s výsledky jeho přepracování do formátu, kterému by PerfEval rozumněl, a znovuuložení do svého adresáře s výsledky měření.

V rámci tohoto řešení je jednoduché zjistit k jaké verzi výsledky měření patří i kde je soubor uložen. Místo uložení souboru je pevně dané jako u prvního způsobu a verze je udaná v metadatech uložených přímo mezi předzpracovanými výsledky. Opět ale dochází k prohledávání souborů na disku jako u první varianty. Aby byl nalezen výsledek měření pro požadovanou verzi bylo by nutné nalézt soubor s výsledky měření k této verzi mezi všemi výsledky měření, které systém eviduje. Lze docílit částečného zrychlení, pokud by se tento systém ukládánní doplnil o nějakou variantu cache. Nicméně při testování dvou verzí z nichž jedna

není uložena v cache by došlo k pomalému prohledávání celé struktury.

Tento způsob tedy Nakonec také nebyl použit, z důvodu pomalého procházení dat uložených v pevné paměti.

Poslední uvažované řešení je strukturované ukládání metadat o výsledcích měření. Využije se databáze o jedné tabulce, kde je uvedena cesta k výsledku měření a verze ke které byl výsledek změřen. Uživatel do systému nahlásí, že má nový výsledek měření, kde je uložen, a k jaké verzi softwaru je měření provedeno. Systém si tato data pouze poznamená do databáze. Při vyhodnocení pak pomocí databázových dotazů nalezen snadno všechny soubory s výsledky měření k potřebné verzi.

Toto řešení poskytuje jak rychlé vkládání nových výsledků, tak rychlé vyhledávání výsledků měření k dané verzi. Zároveň není nutné žádné předzpracování souborů, a tudíž není třeba je při vkládání číst.

4. Zhodnocení řešení

4.1 Ukázka práce

TODO: ukázka práce se systémem a jeho použití

4.2 Nasazení systému v praxi

TODO: ukázka reálného užití systému a jeho výstupu -> evaluate a rúzné výstupy

5. Programátorská dokumentace systému PerfEval

5.1 Architektura systému

5.2 Rozšiřitelnost a její omezení

Systém PerfEval byl od počátku projektován tak, aby byl rozšiřitelný. Hlavní jádro celé aplikace tvoří vyhodnocování výkonu. V různých částech návrhu se vyskytují místa, kde je možné významným způsobem doplnit a změnit chování celé aplikace.

Nejdůležitější ze zmiňovaných rozšíření je rozšíření o datový formát. Tato možnost dělá z PerfEvalu poměrně univerzální nástroj. Činí ho totiž méně závislým na použitém měřícím systému a výstupním jeho formátu.

Rozšiřitelnost systému PerfEval není neomezená. Rozšíření, která nebudou zmíněná v této kapitole, pravděpodobně nebudou možná, nebo budou vyžadovat mnohem více času pro vývoj. Naopak pro rozšíření v této kapitole existuje návod jak systém rozšířit.

5.2.1 Rozšíření o datový formát

Vezměmě opět příklad našeho programátora z první kapitoly. Programátor si napsal výkonnostní testy. Testy napsal pomocí nástroje, který nepodporuje PerfEval. Nicméně progrmátor ví, že systém PerfEval dělá přesně to, co potřebuje. Jediný problém je tedy ve vysvětlení svého datového formátu systému.

PerfEvalje od počátku zamýšlen pro rozšíření v tomto místě. Programátor tedy musí prozkoumat, jak správně zkonstruovat třídu Samples. Třída Samples totiž reprezentuje jedno spuštění testovací metody měřícího frameworku. Všechny hodnoty naměřené jednou metodou jsou tedy uloženy zde.

Programátor musí naimplementovat vlastní implementaci rozhraní MeasurementParser. V tomto rozhraní je důležitá metoda getTestsFromFiles. Tato metoda na vstupu přijme všechny soubory s výsledky měření jedné verze. Výstupem je list objektů typů Samples. Pro každou metodu (měřící test), který se v souborech nachází, se v listu vyskytuje pouze jedna instance typu Samples.

Pokud jsem tedy měření pomocí frameworku spustil dvakrát se stejnými měřícími metodami, tak se ve výsledných Samples každá objeví právě jednou. Naměřené hodnoty budou všechny u této jedné instance Samples.

Poslední krok tohoto rozšíření po naimplementování rozhraní MeasurementParser je jeho registrace. Registrace probíhá tak, že se přidá jeden řádek do statického konstruktoru třídy ParserFactory. Na řádku bude přidání položky do objektu s názvem registeredParsers. Tento objekt je typu HashMap a přiřazuje k sobě Supplier, který vrací MeasurementParser a název typu String. Přidávaná položka je tedy String odpovídající názvu parseru a reference na metodu, která umí parser zkonstruovat.

Použití nového parseru je pak jednoduché. Při inicializaci systému PerfEval příkazem init se jako parametr argumentu benchmark-parser použije jméno no-

vého parseru. Dokonce je začne hlásit v nabídce i varování v případě, že žádný parser není při inicializaci zadán.

5.2.2 Rozšíření o filtr

Pokud uživatel PerfEvalu chce změnit pořadí výpisu testů na výstupu, může implementovat nový filtr. Tento filtr objektem typu Comparator, který porovnává instance MeasurementComparisonRecord. Pomocí tohoto komparátoru, pak dojde k setřídění vypisovaných prvků.

Dále je nutné přepsat metodu resolvePrinterForEvaluateCommand takovým způsobem, aby používala i nový druh filtru. Tuto metodu je možné nalézt ve třídě SetupUtilities. Komparátor se pak může předávat objektům typu ResultPrinter při konstrukci. O jejich dalším použití si tedy tyto objekty rozhodují samy.

5.2.3 Rozšíření o statistický test

Může se stát, že uživatel systému PerfEval má o svých datech nějaké informace, které by chtěl při vyhodnocování zohlednit. Může si tedy naprogramovat vlastní implementaci rozhraní StatisticTest.

Po naprogramování vlastní implementace rozhraní StatisticTest, pak jen stačí ve třídě SetupUtilities změnit chování metody resolveStatisticTest, tak aby brala novou implementaci v potaz. Posledním krokem je přidání nového flagu do parseru příkazové řádky v metodě createParser.

5.2.4 Rozšíření o možnost výpisu

Pro vypisování výsledků porovnání slouží rozhraní ResultPrinter. Pokud by si uživatel chtěl naimplementovat vlastní způsob vypisování, tak stačí impleemtnovat jedoinou jeho metodu PrintResults.

Přidání nového ResultPrinteru je podobné jeko u přidávání nové implementace rozhraní StatisticTest. Stačí změnit implementaci metody resolvePrinterForEvaluateCommand ve třídě SetupUtilities. Úprava by měla být provedena tak, aby metoda uměla vrátit i novou implementaci. Pokud by bylo zapotřebí nového argumentu na příkazové řádce, takje nutné jej také správně naimplementovat do metody createParser.

5.2.5 Změna použitého databázového systému

V důsledku dlouhého rozhodování se o tom, jak se budou informace o výsledcích měření ukládat vzniklo rozhraní Database. Rozhraní má spoustu metod. Pokud by se uživatel rozhodl změnit způsob ukládání dat o měřeních, tak by musel implementovat celé toto rozhraní. Po implementaci rozhraní pak už jen stačí změnit metodu constructDatabase ve třídě SetupUtilities, která vrací instanci objektu typu Database.

5.2.6 Rozšíření o příkaz

Každý příkaz PerfEval se skládá ze dvou tříd. Jedná se o třídu implementující rozhraní CommandSetup a o třídu implementující rozhraní Command. Pokud

by uživatel chtěl doplnit nějaký nový příkaz, který by v kontextu systému dával smysl, tak je to možné. Dobrý příklad pro reprezentování nového příkazu bude vyhodnocení výkonu s grafickým výstupem.

Na rozdíl od stávajícího vyhodnocování by grafické vyhodnocování potřebovalo údaje z více než dvou posledních verzí. Proto by příkaz evaluate-graphical třída rozpoznala jako a vytvořila instanci třídy EvaluateGraphicalSetup. Na této třídě, implementaci CommandSetup, by pak zavolala metodu setup. Metoda setup na třídě EvaluateGraphicalSetup by měla za úkol z konfigurace PerfEvalu a příkazové řádky vyrobit objekt EvaluateGraphicalCommand. Tento objekt, který by implementoval rozhraní Command by pak metoda getCommand na třídě Parser vrátila.

Zbytek programu by se již nezměnil metoda main ve třídě Main by vykonala dodaný příkaz. Spustila by standardním způsobem metodu execute na instanci objektu typu Command.

Zaregistrování nového příkazu by probíhalo přidáním nové položky do statického konstruktoru podobně. Položka mapy commandPerSetup má obdobnou strukturu jako v případě rozšíření o nový MeasurementParser. Dodal by se řádek s názvem příkazu typu String a z reference na bezparametrický konstruktor implementace rozhraní CommandSetup. Název příkazu odpovídá příkazu, kterým se bude volat z příkazové řádky.

5.2.7 Omezená rošiřitelnost ve vyhodnocování

Jeden z nejhorších požadavků na rozšíření systému by bylo rozšíření v oblasti vyhodnocování. Jedná se zejména o změnu implementace třídy PerformanceEvaluator. Tato třída utváří celkovou vyhodnocovací logiku. Její rozšiřitelnost je omezená na impelmentace rozhraní StatisticTest.

Změnu chování vyhodnocování výkonnostních testů lze ale provést. Omezené změny ve vyhodnocování se provádí změnami hodnot uvnitř config.ini souboru.

Závěr

Seznam použité literatury

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha