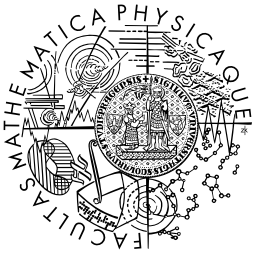

MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

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

Následuje několik ukázkových kapitol, které doporučují, jak by se měla bakalářská práce sázet. Primárně popisují použití TEXové šablony, ale obecné rady poslouží dobře i uživatelům jiných systémů.

1. Kontext

1.1 Testování softwaru při vývoji

Při vývoji softwaru je nutné vyíjený software neustále testovat. Software lze testovat více způsoby, ale cílem testů je vždy otestovat některý z důežitých aspektů jako je například správnost nebo výkon.

Pro testování korektnosti se obvykle používají unit testy. Jedná se o většinou 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 spotřeba paměti. Výsledky měření frameworky umí 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 sady určit něco určitého mohla by být stanovena 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. Frameworky samotné však tuto možnost obvykle nemají.

1.2 Použití automatizovaných nástrojů při vývoji softwaru

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. 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 softeware pomocí unit testů i benchmarků pro měření výkonu. Dále je možné použít jakékoli jiné příkazy příkazové řádky.

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 korektonost již hotového kódu. Dále pojme podezření, že se jeho stávající kód začíná zpomalovat (klesá výkon). Proto se rozhodne použít benchmark 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.

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 sama o sobě nic nevypovídá o změně průběžného výkonu, která je zajímavá. Právě podle změny ve výkonu softwaru jsme schopni 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.

Cílem této práce je vyřešit programátorův problém a vytvořit nástroj, který umožní automatizovaně vyhodnocovat výsledky výkonnostních testů. Použití nástroje by se mělo co nejvíce podobat spouštění a vyhodnocování unit testů. Nástroj by tedy měl umět porovnat strojově zpracovatelné výsledky výkonnostních testů. Na základě statistických metod a jejich parametrů detekovat výrazné zhoršení výkonu a upozornit na něj uživatele. Protože při selhání jednoho z unit testů může být výsledkem selhání celé průběžné integrace, očekává se od požadovaného nástroje podobné chování.

2.2 Použití statistických metod pro analýzu dat

TODO: nalézt nějaký zdroj pro inspiraci

TODO: v následujících sekcích rozebrat jednotlivé použité metody nějak "pořádně".

3. Systém PerfEval

3.1 Popis systému

TODO: Architektura systému, jak systém funguje jako celek a hlavní myšlenky (toku dat)
TODO: Na use case ukázat použití a vysvětlit užití

3.2 Rozbor alternativ v řešení

TODO: grafické rozhraní
TODO: stavba CLI na základě use case
TODO: proč si systém nespoští testy sám
TODO: Proč je poskytováno více statistických metod
TODO: ukládání dat -> DB metadat vs. naparsovaná DB

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

TODO: rozbor rozšíření o statistickou metodu, filtr výsledků, parser vstupních dat, domyslet o co se systém rozšířit nedá

TODO: odkaz na JavaDoc

Závěr

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha