FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Posudek oponenta závěrečné práce

Oponent práce:	Ing. Jan Trávníček, Ph.D.
Student:	Filip Gregor
Název práce:	Výpočet softwarových metrik
Obor / specializace:	Teoretická informatika
Vytvořeno dne:	7. června 2021

Hodnotící kritéria

1. Splnění zadání

- ▸ [1] **zadání splněno**
  [2] zadání splněno s menšími výhradami
  [3] zadání splněno s většími výhradami
  [4] zadání nesplněno

Cílem práce je nastudovat v literatuře zavedené metriky kvality kódu a implementovat nástroj, který bude tyto metriky pro zadaný zdrojový kód počítat. K tomu je přímo zadáním vyžádáno využití kompilátoru Clang. Zadání hodnotím jako středně náročné až náročnější, protože je vyžadováno seznámení s relativně náročným a rozhodně netriviálně velkým nástrojem Clang.

2. Písemná část práce

85 / 100 (B)

Text práce je čtivý, obsah kapitol je relevantní a neshledávám v práci zbytečné části. Ocenil bych kapitolu základních pojmů, která by mohla obsáhnout formální definice grafů a formální úvod do parsování strukturovaného textu.

Faktické poznámky:
Metody v C++ nestačí identifikovat podle názvu a datového typu jejich argumentů, jsou potřeba ještě kvalifikace (například const) a označení && a &.
Metrika ze sekce 2.3.3 je zaměřením jiná než předchozí, a tak by zde mělo dojít k nějakému dělení typu metrik.
Definice 2.4.5 by měla spíše definovat hodnotu metriky SCOM.
Neměl by v popisu sémantického analyzátoru být spíše derivační strom než syntaktický strom?
V příkladu použití třídy RecursiveAstVisitor mi přijde zvláštní, že by existovaly metody WalkUpFromA a WalkUpFromB, protože A a B jsou uvozeny jako třídy v AST, tedy kompilovaném programu, o kterém LLVM obecně nemůže vědět co obsahuje.
V kapitole "Testování" sekci "Počet příkazů" je komentován rozdíl výsledků naměřených

implementovaným nástrojem a nástrojem PMCCabe, nikoli však nástrojem SourceMonitor, který je také v grafu zobrazen.

Jazykové poznámky:
Jména sekcí popisujících jednotlivé metriky jsou někdy pojmenované česky někdy anglicky.
Body 2 a 3 v popisu třídy RecursiveASTVisitor obsahují částečně duplicitu.
Překlep - "Kostra programu tvoří třída".

Typografie:
Anglický abstrakt je nevhodně rozdělen zalomením strany.
Ukázky zdrojových kódů je vhodné opatřit popiskem a vložit do nějakého prostředí. Velmi pravděpodobně by se tím vyřešilo různé zarovnání těchto ukázek (strana 4).
Doporučil bych využívat seskupení textu ve formě příkladu společně s jeho deklarativním uvozením a ukončením pro lepší orientaci v textu (poslední řádek strany 8 navíc uvozuje relativně velký příklad uvnitř paragrafu).
fan-in a fan-out jsou v definici 2.3.6 zřejmě špatně vysázené.
TestWell CMT++ nebo Testwell CMT++?

3. Nepísemná část, přílohy 85 / 100 (B)

Nepísemná příloha ve formě implementace je v jazyce C++. Implementace je otestovaná jednotkovými testy, kterých je dostatečné množství a pokrývají všechny relevantní jazykové konstrukce. Vlastní spuštění nástroje bez parametrů z příkazové řádky způsobí vypsání chyby nedostatku parametrů a pád programu na SIGABRT. Help nástroje (--help) je velmi skoupý na informace, deklarovaný (a funkční) parametr xml v helpu není. Při dodání parametru reprezentujícího cpp soubor k proměření program funguje a vypisuje deklarované informace (softwarové metriky).

Faktické poznámky:
V kódu je zřejmě zbytečná deklarace nečtené proměnné vars v ClassOverviewVisitor::VisitCXXMethodDecl na řádku 44, která je navíc inicializovaná z moved-from položky instance_vars objektu callback.
V implementaci třídy NPathVisitor by bylo vhodnější využít druhého parametru šablony třídy clang::StmtVisitor, deklarujícího návratový typ callback funkcí vzoru visitor, aby si nebylo potřeba komplikovaně a neintuitivně předávat výsledky rekurze pomocí instanční proměnné.
Podmínka if v metodě ClassOverviewVisitor::GetInheritanceChainLen(const std::string &s) const je zbytečná. Typ návratové hodnoty této metody by měl být neznaménkový (podobně i v jiných místech).
Ve třídě FuncInfoVisitor není ideální používat std::pair, navíc je nekonzistentně inicializovaný buď pomocí "list initialization", nebo pomocí make_pair.
stmtClass != Stmt::CompoundStmtClass vs. !(stmtClass == Stmt::CompoundStmtClass) (FuncInfoVisitor.cpp:105).
Metodu StmtNPathVisitor::VisitIfStmt(clang::IfStmt *stmt) je možné zjednodušit prvotním nastavením proměnné result na 1 a efektivně odečtení jedničky v případě existence else větve přesunout až do podmínky.

V kódu se občas vyskytují drobné nekonzistentnosti, například v inicializaci položek třídy je většinou využíváno "list initialization", ale v jednom místě je použito klasické nastavení operátorem =, nebo je občas nekonzistentní obalování/neobalování jednopříkazového bloku uvnitř then větve příkazu if.

4. Hodnocení výsledků, jejich využitelnost 95 /100 (A)

Práci hodnotím jako velmi zajímavou a užitečnou. Žádná softwarová metrika se nemůže rovnat poctivému code review, ale i tak je kvantifikovaná informace ve formě jedné nebo více metrik někdy nezbytná a poskytuje jistou zpětnou vazbu autorovi kódu. Využití nástroje předpokládám cílí na testovač studentských kódů ProgTest, a tam si myslím, že může být v podstatě nasazeno.

Celkové hodnocení 90 /100 (A)

S přihlédnutím k jasné využitelnosti a použitelnosti a i přes výtky k textu a k nástroji samotnému, který jinak zdá se velmi dobře funguje, doporučuji práci k obhajobě a doporučuji ji hodnotit 90 body, tedy známkou A (výborně).

Otázky k obhajobě

V definici metriky Weighted Methods Per Class zmiňujete její vlastnosti při dědění, jsou někde v literatuře popsány její vlastnosti v případě využití kompozice?

Instrukce

Splnění zadání

Posuďte, zda předložená ZP dostatečně a v souladu se zadáním obsahově vymezuje cíle, správně je formuluje a v dostatečné kvalitě naplňuje. V komentáři uveďte body zadání, které nebyly splněny, posuďte závažnost, dopady a případně i příčiny jednotlivých nedostatků. Pokud zadání svou náročností vybočuje ze standardů pro daný typ práce nebo student případně vypracoval ZP nad rámec zadání, popište, jak se to projevilo na požadované kvalitě splnění zadání a jakým způsobem toto ovlivnilo výsledné hodnocení.

Písemná část práce

Zhodnoťte přiměřenost rozsahu předložené ZP vzhledem k obsahu, tj. zda všechny části ZP jsou informačně bohaté a ZP neobsahuje zbytečné části. Dále posuďte, zda předložená ZP je po věcné stránce v pořádku, případně vyskytují-li se v práci věcné chyby nebo nepřesnosti.

Zhodnoťte dále logickou strukturu ZP, návaznosti jednotlivých kapitol a pochopitelnost textu pro čtenáře. Posuďte správnost používání formálních zápisů obsažených v práci. Posuďte typografickou a jazykovou stránku ZP, viz Směrnice děkana č. 26/2017, článek 3.

Posuďte, zda student využil a správně citoval relevantní zdroje. Ověřte, zda jsou všechny převzaté prvky řádně odlišeny od vlastních výsledků, zda nedošlo k porušení citační etiky a zda jsou bibliografické citace úplné a v souladu s citačními zvyklostmi a normami. Zhodnoťte, zda převzatý software a jiná autorská díla, byly v ZP použity v souladu s licenčními podmínkami.

Nepísemná část, přílohy

Dle charakteru práce se případně vyjádřete k nepísemné části ZP. Například: SW dílo – kvalita vytvořeného programu a vhodnost a přiměřenost technologií, které byly využité od vývoje až po nasazení. HW – funkční vzorek – použité technologie a nástroje, Výzkumná a experimentální práce – opakovatelnost experimentů.

Hodnocení výsledků, jejich využitelnost

Dle charakteru práce zhodnoťte možnosti nasazení výsledků práce v praxi nebo uveďte, zda výsledky ZP rozšiřují již publikované známé výsledky nebo přinášející zcela nové poznatky.

Celkové hodnocení

Shrňte stránky ZP, které nejvíce ovlivnily Vaše celkové hodnocení. Celkové hodnocení nemusí být aritmetickým průměrem či jinou hodnotou vypočtenou z hodnocení v předchozích jednotlivých kritériích. Obecně platí, že bezvadně splněné zadání je hodnoceno klasifikačním stupněm A.