BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Předpověď nových chyb pomocí dolování dat v historii výsledků testů

Filip Matys*

Abstrakt
Rozsáhlé open source systémy prochází náročným a často neřízeným vývojem. Jediným způsobem, jak sledovat kvalitu software, je vytváření testovacích sad, jež jsou schopny po provedených změnách objevit softwarové regrese. Tyto testovací sady však rostou společně s vyvíjeným softwarem a testování se tak stává stále náročnější na časové i výpočetní zdroje. Jako řešení tohoto problému se nabízí možnost na základě provedených změn predikovat, které části systému jsou danou změnou ovlivněny a mohou způsobit softwarovou regresi. Díky tomu lze testování soustředit na ohrožená místa a vyhnout se zbytečnému testování míst, jež změnou ovlivněny nejsou. Tento článek popisuje aplikaci, která za pomoci historie výsledků testů a změn kódu získaných z verzovacího systému git tuto funkcionalitu implementuje. Článek popisuje již použité postupy, ale soustředí se hlavně na popis toho, jakým způsobem aplikace doluje potřebná data a dále jakým způsobem jsou tato data využita k predikci softwarových regresí.

Klíčová slova: testování, dolování dat, klasifikace, softwarové regrese

Přiložené materiály: *N/A*

*xmatys10@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Zásahy do softwaru mohou často způsobit, že dojde k softwarovým regresím. Tedy takovým případům, kdy přestane fungovat správně něco, co před zásahem fungovalo. Z tohoto důvodu vývoj software doprovází i tvorba testovacích sad, které mají za úkol různé části systému otestovat a ověřit tak správnou funkcionalitu těchto částí po provedených změnách. S rostoucím software však narůstá i rozsah testovacích sad a s nimi časová, výpočetní a tedy i finanční náročnost samotného testování. Snížení této náročnosti by mohlo být dosaženo, pokud by existovala možnost předpovědi, které části sytému jsou změnou ovlivněny na základě provedených změn, díky čemuž by bylo možné testování na tato místa soustředit a vyhnout se testování místům, jež změnou zasaženy nejsou.

Aby bylo možno předpovídat takováto místa, je potřeba zajistit potřebná data a jejich zpracování. Mezi tato patří historické výsledky testování a změny kódu, jež byli mezi těmito testováními prováděny. Úkolem aplikace je pak vytvořit statistický model, jež je schopen na základě těchto dat predikovat změnu chování části testovací sady. Výstupem predikce je zpravidla odpověď, zda pro testovací sadu hrozí či nehrozí změna chování na základě provedených změn.

Vyvinutá aplikace k predikci používá historická data z nástroje ResultCloud a verzovací systém Git jako zdroj pro získání změn ve zdrojovém kódu. Tato data jsou následně využita k tvorbě statistických modelů pro každý testovací případ z testovací sady a skrze integraci se systémem Git lze pro každou revizi předpovědět, zda provedené změny mají vliv na chování jednotlivých testovacích případů.

Článek popisuje aplikaci sloužící k usnadnění testování rozsáhlých open source systémů, přičemž umožňuje zpracování jakýchkoliv softwarových projektů, které jsou vyvíjeny skrze verzovací systém Git a jejichž výsledky testování jsou importovány do nástroje ResultCloud.

2. Předešlé práce

Na téma predikce softwarových regresí vzniklo množství studií, zabývajícími se jak výběrem vhodných metrik pro popis změn v kódu, tak výběrem ideálního statistického modelu. Výsledky ohledně atributů se od projektu různí, což je potvrzeno i v jedné ze studií. U výběru modelu se však potvrzuje, že nejvhodnějšími modely jsou Naive Bayes a logistická regrese.

Studie Time Menziese [1] zjišťovala, jaký vliv mají na predikci statické metriky, které některé studie považují za málo významné [2][3]. Vstupem této studii byly atributy Halstead [4], cyklická složitost podle McCabe [5] a množství statických atributů. Autoru studie se podařilo vyvrátit tvrzení, že statické metriky mají minimální vliv na předpověď softwarových regresí. Dále ukázala, že atributy s nejvyšším dopadem na předpověď se projekt od projektu liší, jak je ukázáno v tabulce 1, kde jednotlivá čísla odpovídají číslům níže vypsaných atributů. Značení pd představuje pravděpodobnost detekce chyby a značení pf pravděpodobnost, že systém detekuje chybu, avšak při následném testování k chybě nedošlo. Jednotlivé řádky pak představují konkrétní projekty.

Data	pd [%]	pf [%]	Atributy
pc1	48	17	4, 5
mw1	52	15	3, 4
cm1	71	27	2, 4
kc4	79	32	3
pc2	72	14	2, 6
pc3	80	35	1, 5
pc4	98	29	1, 6

Tabulka 1. Přehled výsledků přesnosti pravdivé a nepravdivé predikce regrese v závislosti na datech a vybraných atributech.

1. Počet prázdných řádků
2. Počet řádků komentářů
3. Počet uzlů grafu cyklické složitosti
4. Počet unikátních operandů
5. Počet řádků kódu
6. Procento komentářů

Predikcí se zabývají i ve společnosti Microsoft [6], kde vzniklo několik studií za účelem snížení nákladů na údržbu jejich software. Metriky změny rozdělili na kategorie:

1. **Metriky charakteru opravy.** Popisují význam opravy a její průběh, tedy zda se jedná o opravu nebo novou vlastnost.
2. **Kódové metriky.** Mezi tyto metriky patří kódová složitost, metriky objektově orientovaného charakteru a historie daného kódu.
3. **Metriky závislostí.** Popis závislostí funkcí, modulů, knihoven a datových struktur pomocí grafu závislostí. Ty se rozlišují na externí a interní, respektive na závislosti mezi změněnou a nezměněnou komponentou a závislost mezi dvěma změněnými komponentami.
4. **Zkušenostní metriky.** Popisují změnu z pohledu vývojáře, který změnu provedl. Zejména jeho zkušenost s daným systémem, která je měřena na základě jeho aktivity v posledních několika měsících.

Kategorie	Přesnost odhadu	Směrodatná odchylka
1.	0,73	0,046
2.	0,70	0,040
3.	0,69	0,049
4.	0,54	0,044

Tabulka 2. Výsledky modelu pro různé kategorie metrik

Jak lze z tabulky vyčíst, nejméně informativní kategorií je zkušenost vývojářů, nejvíce pak metriky charakteru opravy.

3. Datové sklady

Datové sklady slouží k uchování dat, u kterých se nepředpokládá, že nad nimi budou prováděny modifikační operace. Často mají časové uspořádání a jednoznačnou identifikaci. Tato data jsou poté použita pro různé analytické účely, v tomto případě pro strojové učení statistických modelů.

3.1 Distribuovaný systém správy verzí

Systém Git se využívá při vývoji softwarových produktů ke správě verzí, je tedy verzovaným datovým skladem. Podporuje týmový vývoj a datové příspěvky od jednotlivých týmových členů řadí v časových řadách do vývojových větví.

Jednotlivé příspěvky, zvané *commity*, jsou identifikovány pomocí hashovací značky. Pomocí příkazů je pak možné přistupovat k jednotlivým verzím.

Git představuje zdroj jak pro samotný kód v různých fázích vývoje, tak pro analýzu změn v kódu, pro kterou má přímou podporu příkazem *diff*, pomocí kterého dojde k zobrazení rozdílů dvou *commitů*, tedy nové soubory, odebrané soubory a změny v již existujících.

3.2 Nástroj pro správu výsledků dlouhodobého testování

ResultCloud vznikl jako webový nástroj pro správu výsledků dlouhodobého testování [7]. Díky své implementaci je schopný za pomoci zásuvných modulů přijímat data různých formátů testovacích sad, která poté ukládá do své interní databáze v jednotné reprezentaci.

Ta je tvořena projektem, jež obsahuje jednotlivé výstupy testovacích sad. Tyto sady jsou pak hierarchicky děleny do kategorií, testovacích případů a následně na jednotlivé výsledky.

4. Návrh aplikace

Aplikace je navržena jako webová aplikace. Návrh se tedy skládá z webového uživatelského rozhraní a poté z části, jež zajišťuje logiku aplikace. Ta se skládá jak z rozhraní, které spojuje uživatelské rozhraní s logickou částí aplikace, tak z jednotlivých služeb, kterými jsou služba repozitáře, služba projektu a služba modelu.

4.1 Uživatelské rozhraní

Uživatelské rozhraní slouží k interakci uživatele s aplikací. Skrze toto rozhraní je možno přistupovat k jednotlivým projektům, jež představují projekty podporující integraci s verzovacím systémem Git v nástroji ResultCloud.

V rámci projektu lze zahájit tvorbu modelů a následně pomocí grafické reprezentace tyto modely procházet. Dále je možné díky integraci se systémem Git procházet jednotlivé revize projektu a v případě existujícího modelu provést nad danou revizí predikci softwarových regresí. Tuto predikci lze zároveň provést pomocí rozhraní pro nahrání souboru na server za předpokladu, že v daném souboru jsou popsány pomocí nástroje diff změny v kódu daného projektu.

4.2 Služba repozitáře

Zajišťuje komunikaci aplikace se systémem Git. Mezi její úkony patří aktualizace lokální kopie na aktuální verzi nebo získání informace o změnách kódu pro danou revizi.

4.3 Služba projektu

Zajišťuje správu projektu v rámci aplikace. Zastřešuje komunikaci s nástrojem ResultCloud, ze kterého získává potřebné informace o projektu. Mezi tyto informace patří samotný seznam dostupných projektů a dále výsledky testování konkrétní revize projektu.

4.4 Služba modelu

Tato služba má na starosti na základě změn v kódu a historii výsledků testování tvorbu modelů a následnou predikci možných softwarových regresí. K tomuto účelu využívá služeb projektu a služeb repozitáře.

5. Implementace aplikace

Tato kapitola popisuje konkrétní řešení návrhu z předchozí kapitoly se zaměřením na tvorbu a použití statistických modelů pro predikci softwarových regresí.

5.1 Analýza změn kódu

Pro analýzu změn kódu je využíván verzovací systém Git, ze kterého jsou pro zvolenou revizi získány informace o změnách v kódu. Tato informace poskytuje názvy souborů, ve kterých se změny odehrály, dále pak výpis řádků, které byli přidány nebo odebrány. Pro každý změněný soubor je vytvořen záznam a tento záznam nese konkrétnější informace o změnách v souboru, které jsou dolovány právě z odebraných a přidaných řádků a mají podobu čítačů, kdy odebrané a přidané změny jsou evidovány. Evidované změny jsou:

- Řádek
- Cyklus
- Změna datového toku
- Přiřazení
- Podmínka
- Příkaz navrácení
- Struktura nebo výčtový typ
- Vazba na externí soubor
- Funkce manipulující s pamětí
- Manipulace s řetězcem
- Multibyte funkce
- Funkce nad řetězcem
- Direktiva
- Pole
- Konstanta
- Specifikátor třídy uložení

Jednotlivé změny jsou nad řádky získávány pomocí regulárních výrazů. Byla snaha i o vytvoření syntaktického stromu nad změněným souborem, což by umožňovalo snadnější sémantickou analýzu. Tento přístup byl však nakonec odložen skrze svou náročnost.

Jelikož cílem práce bylo i zaměřit se na sémantiku konkrétního jazyka, je analýza změn kódu individuální v rámci každého jazyka. Identifikace jazyka v souboru probíhá na základě přípony souboru a jeho podpora je dána existencí zásuvného modulu pro daný jazyk.

5.2 Tvorba modelů

Pro každý projekt jsou jednotlivé modely vytvářeny na základě historie výsledků testování, kdy je pro každý testovací případ iniciován jeden model. Při tvorbě modelu je nejdříve získán seznam všech importovaných

výsledků, následně jsou pak tyto výsledky postupně zpracovávány.

V každé iteraci jsou nejdříve získány výsledky testovacích případů z nástroje ResultCloud, které nesou informaci, zda v daném testovacím případu pro danou revizi došlo ke změně. Tento příznak je pak používán jako atribut pro predikci. Následně jsou z nástroje Git získány informace o změnách kódu dané revize a tyto informace jsou analyzovány způsobem, který je popsán v předchozí kapitole.

Po získání potřebných dat dochází k samotné tvorbě modelů. Pro každý testovací případ jsou získány nejvhodnější atributy pomocí metody Select K-Best, která vybere pro predikci K nejvhodnějších atributů. Následně jsou těmito daty trénovány jednotlivé modely. Prvotní myšlenka byla vytvořit více modelů a tyto modely vyhodnotit a vybrat ten nejlepší. Byl ale nakonec zvolen jiný přístup. Aby bylo docíleno větší přesnosti odhadu změny chování testovacího případu, vytvoří aplikace pro testovací případ sedm modelů pomocí různých metod. Tyto metody jsou:

- Gaussovský Naive Bayes
- Logistická regrese
- SVM
- Gaussovský Naive Bayes s kalibrací sigmoid
- Gaussovský Naive Bayes s kalibrací isotonic
- SVM s kalibrací sigmoid
- SVM s kalibrací isotonic

Zvolené metody jsou standardně využívány pro tvorbu statistických modelů pro předpověď softwarové regrese. Nejvíce používanou metodou, která často dosahuje nejlepších výsledků, je metoda Naive Bayes. Mezi další takovéto metody patří právě metoda logistické regrese a SVM. Ke zmíněným metodám jsou využity i jejich varianty s kalibrací. Kalibrace sigmoid zlepšuje schopnosti metody v případě, kdy by mohlo dojít k přetrénování. Isotonic kalibrace je regresí, která nevytváří jednu lineární funkci jako lineární regrese, ale vytváří na základě vstupních dat množinu vzájemně spojitých funkcí, což v ideálním případě vede ku zpřesnění modelu.

Po vytvoření modelů jsou tyto perzistentně uloženy a tvorba modelů je tím ukončena.

5.3 Predikce

Při požadavku na predikci je pro dané změny provedena analýza a výstup této analýzy je předán veškerým modelům daného projektu. Tyto modely jsou vyhodnoceny a pokud dojde u některého z modelů k predikci, že nastane změna, je testovací případ a výsledky jeho modelů předán uživateli na výstup.

Vzhledem k charakteru vstupních dat nedochází k predikci softwarové regrese, ale k predikci změny chování výstupů testovacích případů. Tento postup byl zvolen z toho důvodu, že data importována do nástroje ResultCloud nemusí představovat veškeré revize projektu a zároveň při predikci softwarové regrese je potřeba znát předchozí výsledky, které nemusí být v daném případě dostupné.

6. Dosažené výsledky

Aplikace byla testována nad open source projektem GDB Binutils. Tento projekt obsahuje několik dílčích nástrojů. Výsledkem testování tohoto projektu je však závěr, že je obtížné pro tak rozsáhlý projekt získat automatizovaně rozsáhlá testovací data, která jsou pro daný projekt relevantní. Rozdělení projektu GDB Binutils na několik dílčích nástrojů totiž způsobilo, že projekt v rámci jednoho repozitáře obsahuje několik dalších na pohled samostatných projektů, které obsahují své vlastní testovací sady. Problém, který zde nastává, jsou jednotlivé revize projektu. Zásahy do projektu totiž nejsou zásahy do projektu jako celku, ale do jednotlivých nástrojů.

Minimální počet vzorků pro statistické učení se zpravidla stanovuje na číslo 30. Za tohoto předpokladu, pokud získám 30 různých revizí projektu GDB Binutils a zpracovávám výsledky testování těchto revizí pro konkrétní nástroj tohoto projektu, tak můžu předpokládat, že ani jeden ze zásahů do kódu zachycených v daných revizích není zásahem do zvoleného nástroje. Podobný předpoklad mohu mít pro libovolný počet revizí, který je větší jak nula a menší jak jejich celkový počet.

Aby tedy aplikace byla schopna vytvořit modely, jež je možné prakticky využít k predikci změn výstupu testovacích případů, je potřeba nástroj ResultCloud plnit daty, jež jsou relevantní k danému nástroji v případě, že jsou jednotlivé nástroje v projektu nezávislé, nebo zajistit jejich dostatečné množství, které však nelze jednoznačně předem určit.

Příkladem zmíněných závěrů jsou modely testovacích případů, které dosahují 100% úspěšnosti, což bylo způsobeno absencí případu, kdy v daném testovacím případu napříč zkoumanými revizemi došlo k nějaké změně. Pokud k nějaké změně došlo, tak se zpravidla jednalo o změnu pouze jednu, což má v konečném důsledku minimální vypovídající hodnotu.

7. Závěr

Aplikace implementuje požadovanou funkcionalitu, její testování je však na vybraném vzorku dat problematické. Projekt GDB Binutils byl vybrán z důvodu své

dostupnosti skrze verzovací systém Git a také kvůli formátu testovací sady, kterou je DejaGnu, jež je podporována nástrojem ResultCloud.

Budoucnost aplikace závisí na dalších studiích, které se budou zaměřovat zejména na zpětné získání relevantních dat pro tvorbu použitelných modelů.

Literatura

[1] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[2] Norman E. Fenton and Ieee Computer Society. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. Softw. Engng*, pages 797–814, 2000.

[3] K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort. *IEEE Trans. Softw. Engng*, pages 126–137, 1995.

[4] M. H Halstead. Elements of software science. 1977.

[5] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[6] Alexander Tarvo. Using statistical models to predict software regressions. In *Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, ISSRE '08, pages 259–264, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Filip Matys. Webový nástroj pro správu výsledků dlouhodobého testování. Technical report, Vysoké učení technické v Brně, 2014.