VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZPOZNÁNÍ PLAGIÁTŮ ZDROJOVÉHO KÓDU V JAZYCE PHP

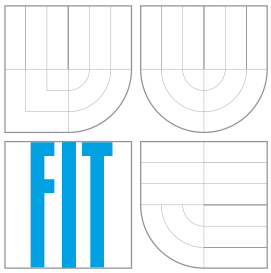
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ KRPEC

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZPOZNÁNÍ PLAGIÁTŮ ZDROJOVÉHO KÓDU V JAZYCE PHP

PLAGIARISM RECOGNIZER IN PHP SOURCE CODE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ KRPEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZBYNĚK KŘIVKA, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Cílem práce je vytvořit aplikaci, která rozpozná plagiáty projektů napsaných v jazyce PHP. Za plagiátorství lze považovat úmyslné kopírování cizího kódu, případně jeho transformací a jeho vydávání za vlastní.

Abstract

Main goal of this thesis is to create application, which can detect plagiarism in source code written in PHP language. Plagiarism is viewed as a form of code obfuscation where plagiarists deliberately perform semantics preserving transformations of original version to pass it off as their own.

Klíčová slova

PHP, plagiát, odhalování plagiátů, Halsteadova metrika

Keywords

PHP, plagiarism, plagiarism detection, Halstead metric

Citace

Ondřej Krpec: Rozpoznání plagiátů zdrojového kódu v jazyce PHP, bakalářská práce, Brno, FIT VUT v Brně, 2015

Rozpoznání plagiátů zdrojového kódu v jazyce PHP

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D.

.....
Ondřej Krpec
19. ledna 2015

Poděkování

Velmi rád bych poděkoval vedoucímu mé bakalářské práce Ing. Zbyňku Křivkovy, Ph.D. za jeho připomínky, odborné rady, čas a ochotu při konzultacích.

Obsah

Kapitola 1

Úvod

Plagiátorství je závažným problémem nejen ve vzdělávacích a vědeckých institucích a má velmi dlouhou a bohatou historii. Mezinárodní norma ČSN ISO 5127-2003 jej popisuje jako představení duševního díla jiného autora, půjčeného nebo napodobeného vcelku nebo zčásti, jako svého vlastního. Nicméně nejvíce případů plagiátorství se stále nachází v akademických institucích, kde studenti kopírují své práce mezi sebou, v přesvědčení, že zahladili všechny stopy vedoucí k odhalení plagiátu.

Pokud se zaměříme čistě na plagiáty ve zdrojových kódech, můžeme je definovat jako program, který byl vytvořený z jiného programu tak, aby na první pohled nebylo možné rozeznat originál od kopie. Mezi základní transformace zdrojového kódu, tedy ty, na které se tato práce zaměřuje můžeme označit změnu komentářů, identifikátorů, řídících struktur, restrukturalizaci zdrojového kódu nebo změnu toku programu (výměna podmínek ve vyhodnocení `if-else`).

V rámci této bakalářské práce se pokusím ukázat způsoby, jak odhalit právě plagiáty studentských prací napsaných v jazyce PHP a poukázat na problémy, ke kterým může při automatické detekci plagiátorství docházet.

1.1 Plagiátorství

Jak již bylo uvedeno výše, plagiátorství je na akademické půdě závažným problémem a tudíž by mělo být i příslušně potrestáno. Nicméně plagiátorství zdrojových kódu sebou přináší několik problému, které značně stěžují schopnost odlišit plagiát od originálu. Tyto problémy můžeme rozlišit do šesti kategorií (Figure 1.1), z nichž pouze jednu můžeme označit jako plagiát.

- **Zdrojový kód třetích stran**, kterým jsou myšleny různé open-source kódy případně různé knihovny.
- **Nástroje na automatické generování kódu**, kde jako příklad můžeme uvést vývojové prostředí Netbeans IDE, které automaticky vytváří některé metody.
- **Obvykle používané identifikátory** jako například proměnné *result* nebo *i*.
- **Obvykle používané algoritmy** budou ve většině připadech implementované stejným způsobem. Jako příklad zde lze uvést téměř libovolný řadící algoritmus.

- **Společný autor** jednoho nebo více programů může vytvořit více různých verzí programů, které se mohou jevit jako plagiáty, protože autor má tendenci psát kód svým naučeným způsobem.
- **Opsaný kód**, který jako jediný může být označet jako plagiát, protože zde došlo ke kopírování nebo transformaci cizího kódu bez patřičného uvedení jeho autora.

Figure 1.1: Zeidmanův seznam šesti kategorií podobností kódu [odkaz na Bob Zeidman. What, exactly, is software plagiarism? Intellectual Property Today, 2007.][1]

Detekce plagiátorství na akademické úrovni sebou bohužel přináší ještě další problémy. Zadávané úkoly, hlavně ty v začatecnických kurzech programování bývají standardizované a velmi striktně zadávané, což může vyústit v podobně napsané programy, přestože studenti vypracovávali zadaný úkol samostatně.

1.2 Typy plagiátorství

Jako plagiátorství označujeme nejen jednoduché zkopírování zdrojového kódu, ale také jeho transformace. Tyto mohou být velice jednoduché jako pouhá změna, odebrání nebo přidávání komentářů, případně přejmenováním proměnných, ale mohou být také komplikovanější jako třeba změna struktury kódu tj. různé vnořování funkcí nebo přepsání `for` cyklů na `do-while` cykly. Případně je možné, že se autor může pokusit využít modularity. Modularita nebo také rozčlenění programu do několika spolupracujících modulů (souborů) je při práci na rozsáhlejších projektech samozřejmostí. Tento prostředek nicméně umožňuje plagiátorovi vhodně schovat opsaný kód tak, že prohodí funkce mezi jednotlivými moduly. J. A. W. Faidhi a S. K. Robinson podle těchto předpokladů definovali šest úrovní plagiátorství zdrojového kódu (Figure 1.2) od nejjednodušších technik až po ty nejsložitější.

- **Úroveň 1** - změna komentářů ve zdrojovém kódu
- **Úroveň 2** - změna názvu identifikátorů
- **Úroveň 3** - změna pozice proměnných ve zdrojovém kódu
- **Úroveň 4** - změna konstant a funkcí
- **Úroveň 5** - změna cyklů
- **Úroveň 6** - změna struktur určených pro kontrolu toku programu

Figure 1.2: Faidhi a Robinson definovali šest úrovní plagiátorství zdrojového kódu.[J. A. Faidhi and S. K. Robinson. An empirical approach for detecting program similarity and plagiarism within a university programming environment. 1987.]

Tato práce je zaměřena na odhalení všech šesti úrovní plagiátorství, přičemž hlavní pozornost je věnována prvním čtyřem úrovním, které by měly být odhaleny vždy již při povrchním porovnávání zdrojových kódů.

1.3 Jazyk PHP

Jazyk PHP se ve své první formě objevil již v roce 1994, kdy se pan Rasmus Lerdorf rozhodl, že vytvoří jednoduchý systém, který bude započítávat přístup na webové stránky. První verze byla napsána v jazyce PERL, nicméně vzhledem ke značnému zatížení serveru bylo poté PHP přepsáno do jazyka C. V průběhu let následovalo vydání několika dalších verzí, až nakonec v roce 2003 byla oficiálně vydána beta verze PHP5, která přinesla největší změnu v podobě přidání objektového modelu.

Jazyk získal velké uplatnění zejména z důvodu, že je nezávislý na platformě a rozdíly v různých operačních systémech se omezují pouze na několik systémově závislých funkcí. Nicméně stejně jako ostatní jazyky, má i PHP nevýhody. Největší nevýhodou je fakt, že se jedná o jazyk interpretovaný, což znamená, že při jakémkoliv spuštění i toho nejmenšího skriptu, je potřeba soubor s tímto skriptem znovu kompilovat.

1.3.1 Specifika jazyka PHP

Jak již bylo uvedeno výše, tak objektový model byl do jazyka přidán až později, což nyní programátorům umožňuje vybrat si, jestli budou své programy psát využívajíc imperativního nebo objektově orientovaného paradigmatu. Tento výběr ovšem také napomáhá šíření plagiátorství, protože lze přepsáním originálního zdrojového kódu do jiného programovacího paradigmatu vytvořit na první pohled odlišný kód.

1.3.2 Kvalita kódu

Jazyk PHP byl dlouho definován pouze svou implementací a oficiální specifikace jazyka byla oznámena teprve na konci července 2014. Pokud k tomuto připočteme fakt, že v jazyku panuje nekonzistentní pojmenování funkcí, případně nejednotné pořadí parametrů ve funkcích a vezmeme v potaz i to, že jazyk samotný není obtížné se naučit, dostaneme jako výsledek mnoho nekvalitního kódu.

Kvalitu zdrojového kódu lze podle Dr. Billa Curtise definovat jako pětici požadovaných vlastností softwaru, nutných k zajištění jeho obchodní hodnoty. Tento model byl později uveden jako CISQ model kvality. (Figure 1.3)

- **Spolehlivost** - Určuje míru rizika a pravděpodobnosti případných selhání aplikace.
- **Efektivita** - Zdrojový kód a jeho struktura jsou hlavními faktory, které určují rychlost běhu aplikace.
- **Bezpečnost** - Metrika na určení náchylnosti programu na prolomení bezpečnosti zapříčiněné špatnými praktikami v návrhu a naprogramování aplikace.
- **Udržovatelnost** - Zahrnuje srozumitelnost a přenositelnost aplikace, jak mezi jednotlivými operačními systémy, tak mezi jednotlivými vývojovými týmy.
- **Velikost** - Ačkoliv velikost přímo neovlivňuje kvalitu zdrojového kódu, tak stále má velký vliv na udržovatelnost kódu a tedy nepřímo i na kvalitu výsledného kódu.

Figure 1.3: Definice kvality zdrojového kódu podle CISQ modelu[link na cisq]

Kapitola 2

Analýza problému

Jak již bylo naznačeno v úvodu, plagiát zdrojového kódu můžeme definovat jako program, který byl vytvořen z jiného programu s určitým počtem transformací tak, aby na první pohled nebylo možné poznat zdroj. Z toho vyplývá velmi důležitý požadavek na vlastnost výsledného programu, tedy schopnost rozeznat plagiáty mezi projekty, které jsou pouze podobné, ale ne opsané. K zajištění této vlastnosti, bylo využito několika rozdílných technik, které budou popsány níže, tak aby program nalezl a vyznačil plagiáty co možná nejpřesněji a zároveň neoznačil projekty, které byly pouze v některých ohledech podobné.

Základem při využívání jakýchkoliv z následujících technik pro rozpoznání plagiátů je pokus o zredukování co možná největšího počtu nadbytečných znaků, které samotné nemají vliv na samotný běh programu. O toto se postará lexikální analýza při překladu zdrojového kódu, nicméně vzhledem k povaze našeho programu nejsou zdrojové kódy překládany a spouštěny a je nutno zajistit některé funkce lexikální analýzy externě. První důležitou funkcí, kterou musí lexikální analýza pro potřeby usnadnění detekce plagiátů je odstranění jakýchkoliv bílých znaků ze zdrojového textu. Druhou funkcí je transformovat zdrojový kód do posloupnosti tokenů.

2.1 Tokenizace

Jedná se o proces, který provádí lexikální analyzátor při překladu programu. V jeho průběhu se načítají ze vstupního zdrojového souboru znaky reprezentující zdrojový program a z těchto znaků jsou následně vytvořeny symboly programu tzv. lexémy. Tyto lexémy jsou následně reprezentovány ve formě tokenů, což je řetězec jednoho nebo více znaků, které jsou v daném jazdyce důležité jako skupina[odkaz na list of parser tokens php].

Využití tokenů při porovnávání zdrojových kódu má oproti porovnání přímo zdrojových textů velikou výhodu v tom, že každý token má v programu specifický význam a také hodnotu původního řetězce. Díky využití této abstrakce, kdy se pomocí metod na detekci plagiátorství hledají plagiáty, lze již předem zajistit, aby pouhé přejmenování proměnných a funkcí v opsaném kódu neovlivnilo činnost těchto metod. Výslednou posloupnost tokenů lze dále použít jako vstup pro některé techniky na detekci plagiátů, které budou uvedeny níže. Další možností je z tokenů vytvořit abstraktní syntaktický strom.

2.2 Abstraktní syntaktický strom

Jedná se o stromovou reprezentaci abstraktní syntaktické struktury zdrojového kódu napsaného v programovacím jazyce. Struktura tohoto stromu je založena na principu, že vnitřní uzly stromu jsou operátory a listy stromu jsou operandy. Tohoto stromu se v překladačích využívá zejména pro překlad a optimalizaci kódu nicméně díky tomu, že každá část podstromu je samostatnou logickou jednotkou lze pomocí algoritmů pro porovnávání stromových struktur zjistit podobnosti zdrojových kódu i pokud byl plagiát různě restrukturalizován. Samotná syntaxe stromu je částečně abstraktní, což znamená, že přímo nereprezentuje každý detail, který se vyskytuje v reálné syntaxi. Jako příklad lze uvést složené závorky, které v jazyce PHP obalují syntaktické konstukce větvení `if-else`, které může být ve stromu reprezentováno pouze jediným uzlem se dvěma větvemi.

TODO: Obrázek AST pro eukliduv algoritmus

Na obrázku 2.2.1[link] je předvedena ukázka, jak vypadá abstraktní syntaktický strom pro Eukleidův algoritmus. Na obrázku lze výborně vidět jistou míru abstrakce, kterou tyto stromy poskytytují.

```
function Euklid(int a, int b) {
   while b != 0 {
      if (a > b) {
         a = a - b;
      }
      else {
         b = b - a;
      }
   }
   return a;
}
```

Zdrojový kód pro Euklidův algoritmus na výpočet největšího společného dělitele.

2.3 Halsteadova metrika

Halsteadova metrika velikosti programu je softwarová metrika, kterou v roce 1977 představil Maurice H. Halstead. Je založena na předpokladu, že všechny programy se skládají z konečného počtu programových jednotek, tzv. tokenů, které jsou rozeznatelné v syntaktické fázi překladačem. Počítačový program poté může být brán jako posloupnost těchto tokenů, které mohou být dále klasifikovány buď jako operátory nebo operandy. Cílem techniky bylo identifikovat takové vlastnosti programů, které by byly snadno vyčíslitelné, a které by mezi sebou měli určité souvislosti. Proto Halstead pro svoji metriku definoval čtyři základní proměnné, ze kterých lze poté vypočítat další metriky programu.

- η_1 = počet unikátních operátorů
- η_2 = počet unikátních operandů

- $\mathbf{N}_1$ = celkový počet operátorů
- $\mathbf{N}_2$ = celkový počet operandů

Z těchto čisel lze dále vypočítat hned několik základních metrik.

- **Délku slovníku:** $\eta = \eta_1 + \eta_2$
- **Délku programu:** $N = N_1 + N_2$
- **Odhadnutou délku programu:** $N = \eta_1 log\eta_1 + \eta_2 log\eta_2$
- **Objem:** $V = N * log_2\eta$
- **Programovou náročnost:** $D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2}$
- **Programátorské úsilí:** $E = D * V$

Uvažujme následující funkci v jazyce PHP:

```
function getAverage() {
fscanf(STDIN, "%d %d %d", $a, $b, $c);
$avg = ($a + $b + $c) / 3;
echo "avg = " . $avg;
}
```

Unikátní operátory jsou následující: `'fscanf'`, `'('`, `'STDIN'`, `'%d %d %d'`, `'$a'`, `'$b'`, `'$c'`, `')'`, `';'`, `'$avg'`, `'3'`, `'echo'`, `'avg = '`
Unikátní operandy jsou následující: `','`, `'='`, `'+'`, `'/'`, `'.'`
$\eta_1 = 13$, $\eta_2 = 5$
$N_1 = 20$, $N_2 = 9$
Délka slovníku:
Délka programu:
Odhadnutá délka programu:
Objem:
Programová náročnost:
Programátorské úsilí:

Příklad 2.1.1

Výhodou použití této techniky k detekci plagiátorství je její rychlost. Halsteadova metrika pouze vypočítá výše uvedené metriky pro určitý zadaný úsek kódu a poté stači tyto hodnoty porovnat s metrikami získanými z jiných zdrojových kódu. Její velkou nevýhodou je nicméně fakt, že je možné získat dvě stejné metriky pro zcela rozdílně pracující kusy kódu, které by byly následně vyhodoceny jako plagiát. Proto byla tato technika využita pouze jako odrazový můstek pro časově náročnější techniky, aby se tyto mohly zabývat pouze projekty, u kterých jsou předpoklady k nalezení opsaných částí zdrojového kódu.

2.4 Levenshteinova vzdálenost

Jedná se o jednu ze základních metrik k detekci plagiátorství textů zavedena v roce 1965 Vladimírem Levenshteinem.Vzdálenost dvou řetězců je zde definována jako minimální počet operací vkládání, mazání a substituce takových, aby po jejich provedení byly zadané řetězce totožné. Samotný zdrojový kód je pak pouze posloupnost řetězců, takže není žádný problém tento algoritmus aplikovat na dva různé zdrojové kódy a rozhodnout, jak jsou si podle tohoto algoritmu navzájem podobné.

Jako příklad Levenshteinovy vzdálenosti lze uvést třeba anglická slova `house` a `router`. Jejich vzdálenost je tři, protože minimální počet operací, které je potřeba provést k transformaci jednoho slova na druhé je právě tři. Pokud bychom chtěli transformovat slovo `house` na `router`, stačí pouze ve slově `house` substituovat `h` za `r`, `t` za `s` a nakonec pouze přidat na konec slova znak `r`.

2.5 Otisky dokumentů

Patří mezi jednu z nejpoužívanějších metod pro detekci plagiátorství, kterou využívají i některé akademické instituce. Metoda funguje na principu identifikování některých specifických řetětců v dokumentu a z nich vytvoření unikátního otisku. Teoreticky by tak rozdílné dokumenty měly vždy mít rozdílné otisky a stejně tak, podobné dokumenty by měly mít otisky podobné.

Většina metod na tvorbu otisků využívá principu tzn. *k-gramů*, kde *k-gram* je sousedící podřetězec o délce *k*. Kód níže demostruje příklad tvorby takového otisku dokumentu, kdy nejprve jsou odstraněny ze zdrojového kódu bílé znaky, posléze jsou vytvořeny 5-gramy, na ně aplikována hashovací funkce a nakonec je z výsledku této hashovací funkce podle určeného vzoru vybrán vzorek dat.

- `public static String s = "Hello";`
- publicstaticStrings=”Hello”;
- publi ublic blics licst icsta cstat stati tatic aticS ticSt icStr cStri Strin tring rings ings= ngs=”gs=”H s=”He =”Hel ”Hell Hello ello”llo”;
- 10701 11107 9382 10239 10003 9496 10975 11013 9314 11035 10000 9401 8022 11062 10851 10851 10034 10476 9860 10805 5742 3364 6960 9660 10306
- 10701 10003 9314 8022 10034 5742 10306

Figure 2.5.1 Otisk vzorového dokumentu

Nevýhodou celého řešení je bohužel to, že *k-gram* sdílený dvěma dokumenty je nalezen pouze v případě, že je vybrán do kroku číslo pět, jak je vidět na obrázku 2.5.1[link] K překonání tohoto problému se využívá algoritmus pro výběr otisků z k-*k-gramů*, který z množiny otisků vybere několik reprezentativních vzorků tj. skupinu otisků rozdělí na určité úseky tzn. okna, které se nepřejkrývají. Z každého úseku je poté vybrán nejmenší otisk.

Princip takovéto detekce plagiatorství spočívá v lokálním charakteru tvorby otisků. Zdrojový kód ovlivňuje jejich výběr a je zaručeno stejné zpracování pro různé dokumenty nehledě na to, zda se podobný kód objeví na každém desátém řádku, pouze na začátku souboru nebo náhodně. Vždy pokud se ve dvou dokumentech vyskytuje stejná posloupnost tokenů, z obou dvou bude vybrán stejný vzorek.

Kapitola 3

Návrh řešení

Po analýze daného problému bych se v následující kapitole dál věnoval jeho samotnému řešení. Podrobně bude popsána každá fáze, kterou budou zdrojové texty procházet. Celý program byl rozdělen do několika fází, které na sebe postupně navazují, nicméně každá z nich má svůj samostatný výstup, který lze v některých případech použít jako vstup pro následující fáze programu.

Před samotným započetím samotné detekce je zapotřebí připravit projekty tak, aby bylo umožněno jednoduché porovnávání zdrojových textů. Jak již bylo uvedeno v kapitole 1.2 [ ] je potřeba zabránit schovávání opsaných částí kódu do modulů. Vzhledem k tomu, že výsledný kód nemusí být pro potřeby samotné detekce přeložitelný, lze před započetím prací spojit všechny moduly z projektů pomocí skriptu do jednoho. Tento jeden soubor je poté připraven ke zpracování pomocí lexikální analýzy.

3.1 Lexikální analýza

Lexikální analýza je činnost, kterou provádí tzv. lexikální analyzátor a v jejímž průběhu se načítají ze vstupního zdrojového souboru znaky reprezentující zdrojový program a z těchto znaků jsou následně vytvořeny symboly programu tzv. lexémy. Tyto lexémy jsou následně reprezentovány ve formě tokenů, což je řetězec jednoho nebo více znaků, které jsou v daném jazyce důležité jako skupina. Tento proces se nazývá tokenizace. V průběhu tokenizace je také pro zajištění rychlejší a přesnější kontroly plagiátů potřeba odstranit ze zdrojového textu veškeré bílé znaky. Jakmile je tento proces ukončen, posloupnost tokenů je serializována do formátu JSON a uložena do souboru pro další použití.

3.2 Analýza tokenů

V momentě kdy je vstupní soubor převeden do posloupnosti tokenů a serializován do struktury JSON, lze začít s jeho analýzou. Tato fáze probíhá v několika krocích, které ovšem v rámci efektivnosti algoritmu jsou prováděny současně. V průbehu analýzy je vhodné

3.3 Halsteadova metrika

3.4 Levenstheinova vzdálenost

3.5 Porovnání abstraktních syntaktických stromů

3.6 Metoda winnowing

3.7 Řízení programu

Kapitola 4

Experimenty

Kapitola 5

Výsledky a zhodnocení práce

5.1 Testování a statistiky

5.2 Návrhy na rozšíření

Kapitola 6

Závěr

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

Literatura

[1] Kolektiv autorů: *Pravidla českého pravopisu*. Academia, 2005, iSBN 80-200-1327-X.