Počítanie riadkov zdrojového kódu

MILAN ŽUŽO

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava

Abstrakt. V súčasnosti sa metóda počítania riadkov zdrojového kódu bežne používa na odhadovanie času (najčastejšie v jednotke človeko-mesiace) potrebného na úspešné dokončenie projektu. Taktiež sa využíva na meranie efektivity práce programátora alebo tímu. V tejto práci opisujem rôzne pohľady na počítanie riadkov zdrojového kódu. Zameriavam sa hlavne na problémy spojené s používaním tejto metódy v spomenutých dvoch oblastiach a podrobnejšie rozoberám rôzne prístupy k počítaniu týchto riadkov. Podávam náčrt alternatívnych možností merania produktivity. Každá metóda má nejaké nedostatky a preto je mojim cieľom v tejto práci popísať spôsob, akým sa dá táto metóda používať vzhľadom na jej nedostatky.

Úvod

Počítanie riadkov zdrojového kódu (LOC – lines of code) je jedna z najstarších, ak nie práve najstaršia metóda merania veľkosti softvérových produktov. Aj napriek tomuto faktu sa ešte stále bežne používa na odhadovanie veľkosti projektu alebo na meranie produktivity práce či už programátora alebo celého tímu. Nastáva otázka, či je táto metóda stále použiteľná. Vývoj softvéru predsa prešiel výraznými zmenami. Má riadok kódu v asembleri, ktorý sa písal pred niekoľkými rokmi, porovnateľnú hodnotu s riadkom v niektorom z vyšších programovacích jazykov? Jedným z riešení je napríklad prevodová tabuľka, ale je jasné, že pri používaní tejto metódy vo vyvíjajúcom sa prostredí nastanú isté komplikácie. Takéto a podobné problémy viedli k vzniku nových metód, ako je napríklad metóda funkčných bodov (angl. Function Points). Aj keď sa snažia nahradiť počítanie riadkov zdrojového kódu, stále nie sú univerzálne použiteľnými metódami – majú svoje výhody, ale aj nevýhody. V tomto článku sa pokúsim priblížiť problémy, ktoré vznikajú pri počítaní LOC a prípadne spôsoby odstránenia ich vplyvu.

Manažment v softvérovom inžinierstve, máj 2005, s. 1-6.

Odhad veľkosti projektu

V dnešnej dobe sa projekt robí pre zákazníka a ten chce vedieť, kedy bude produkt hotový. Konkurencieschopnosť firmy závisí od schopnosti dodať výrobok včas a v požadovanej kvalite. Preto sa manažment, ktorý je zodpovedný za daný projekt, musí zaoberať problémom stanovenia termínu dokončenia projektu. Tieto odhady sa opierajú o metódy, ktoré merajú veľkosti projektov a snahou je aproximovať podobné projekty na náš konkrétny pripravovaný projekt. Ale aj v súčasnosti používané metódy nie sú bezchybné – vždy je to len odhad.

LOC

Ako Phillip Armour popisuje vo svojich článkoch [2] a [3], použitie počítania LOC na odhadovanie času potrebného na úspešné dokončenie projektu nás privádza k dileme. Keď chceme odhadovať veľkosť projektu, tak sme ešte len v počiatočnom štádiu projektu a nemáme ešte žiaden kód – teda nemáme čo počítať. A keď už máme kód, tak je projekt už hotový a nepotrebujeme odhadovať. Takže to vyzerá tak, že LOC pri odhadovaní sú presné, ak ich nepotrebujeme a nie sú k dispozícii, ak ich potrebujeme.

Hlavným problémom je to, že naším cieľom nie je získať LOC ale vedomosti. Ak riadky zdrojového kódu neobsahujú tie „správne“ vedomosti, je jedno koľko ich je. Oveľa horší je fakt, že nepotrebujeme vedieť množstvo vedomostí, ktoré máme, ale to, ktoré musíme získať (ktoré nám chýba, aby bol projekt hotový). Ľudia ešte stále nevymysleli jednotku pre množstvo vedomostí ani spôsob ako ich odvážiť, takže takéto odhady nie je ľahké urobiť. LOC možno vyjadrí nejaký priemer, ale použiť ju ako jednotku na meranie množstva vedomostí je skoro ako odváženie knihy za účelom zistenia, koľko vedomostí obsahuje.

Projekt však nepozostáva iba zo zdrojového kódu. Na jeho zvládnutie treba prejsť rôznymi fázami jeho životného cyklu, ako je zber požiadaviek, analýza, návrh, testovanie a pod. Niekedy sa požiadavky zbierajú týždeň, inokedy dva mesiace. A táto informácia sa v počte riadkov zdrojového textu programu nevyskytuje.

Funkčné body

Štandardný prístup International Function Points User Group (IFPUG) zahŕňa počítanie a váženie vstupov, výstupov a základných prvkov na uchovávanie dát s úpravami zvolenými s ohľadom na prostredie, v ktorom bude systém pracovať [1]. Tieto prvky sú skoro vždy prístupné v skorej časti projektu, keď je potrebné spraviť odhad. Ale pri tejto metóde môžme pozorovať zopár zaujímavostí. Napr. bežným krokom procedúry metódy funkčných bodov je preklad medzi LOC a funkčnými bodmi prostredníctvom procesu, ktorý sa nazýva „backfiring“ [4]. Ak vezmeme do úvahy, že pôvodným zámerom bolo odísť od LOC, pôsobí to trochu ironicky. Niektoré procedúry odhadovania umožňujú merať veľkosť systému podľa modulov, komponentov, objektov a iných častí systému. Ale vo väčšine prípadov musia byť tieto počty sprevádzané faktorom, ktorý určuje, koľko LOC je na daný prvok.

Ak odhliadneme od toho, že niektoré procedúry sa vracajú k LOC, vyzerajú funkčné body na odhadovanie veľkosti projektu o trocha lepšie ako samotné počítanie LOC. Ale aj tu nastáva problém s tým, že podľa Phillipa Armoura nemeriame vedomosti. Napr. zložitosť telekomunikačného systému nemusí vôbec súvisieť s jeho vstupmi a výstupmi. V takomto prípade nedostaneme správny odhad.

Podrobnejší opis iných metód ako LOC je mimo rozsah tohto článku, preto sa im nebudem venovať. Chcel som iba ukázať, že aj keď funkčné body pôvodne chceli nahradiť počítanie LOC (keďže počet riadkov ešte na začiatku projektu nemáme), nakoniec sa aj tak spravil prevod medzi týmito dvoma metódami.

Meranie produktivity práce

LOC sa často používa na meranie produktivity práce zamestnancov. Nemusí to byť vždy za účelom zisťovania, kto ako pracuje, ale taktiež sa touto metódou dá sledovať pokrok projektu. Ak sa výrazne zmení počet LOC za dané skúmané obdobie oproti predošlým obdobiam, je to indikáciou toho, že niečo možno nie je v poriadku a manažment projektu má šancu včas zareagovať a identifikovať problém.

Veľa programátorov má však voči tejto metóde predsudky, pretože zastávajú názor, že nie je objektívna. Zavedenie tejto metódy do takéhoto prostredia môže byť niekedy kontraproduktívne. Na jednom fóre som sa stretol s prípadom, kedy manažér zaviedol do tímu počítanie LOC. Programátori vtedy začali rozťahovať formátovanie kódu a písať zbytočne rozsiahle komentáre. Keď na to manažér prišiel a začal počítať bodkočiarky (písali vtedy v jazyku C), programátori začali dávať bodkočiarky do komentárov. Možno ich prístup nebol veľmi profesionálny a vzťah manažéra s tímom možno tiež nebol najlepší, ale vyplýva z toho isté ponaučenie. Nie je vždy ľahké zaviesť do praxe metódu, o ktorej nie je každý presvedčený, že je objektívna.

Meranie produktivity so sebou nesie aj iné komplikácie. Občas sa optimalizáciou zníži počet riadkov kódu. Vezmime si situáciu, kedy programátor potrebuje istú časť kódu optimalizovať a zaberie mu to celý deň. Na konci dňa úspešne dokončí svoju prácu, ale má menej riadkov kódu ako na začiatku dňa. Nemôžme predsa povedať, že nepracoval. Preto výkyvy v počte riadkov treba zaznamenávať s poznámkami o tom, čo bolo príčinou daného poklesu. Možno lepším spôsobom by bolo počítanie riadkov, ktoré sú odlišné od napr. predchádzajúceho dňa. Takáto metóda by zahrnula aj optimalizáciu kódu, či opravovanie chýb. Na druhej strane sa to dá takisto zneužiť lenivými členmi tímu. Ale pri správnej aplikácii do tímu by táto metóda bola pravdepodobne o niečo presnejšia ako počítanie LOC.

Ďalšou možnosťou pre sledovanie pokroku tímu je percentuálne vyjadrenie toho, koľko tím z rôznych súčastí svojej práce/projektu dokončil. Či sa jedná o vylepšovanie jednotlivých častí programu (napr. COM alebo DLL) alebo celý program, vývojár má mapu (aspoň vo svojej hlave) toho, čo treba urobiť, ako je to rozdelené do menších častí, ako zložitá môže daná časť byť, v akom časovom rozmedzí by sa daná časť dala dokončiť, atď. S realizáciou je to už ťažšie, pretože takýto odhad spraví človek najlepšie iba sám pre seba. Na posúdenie schopností tímu treba nejakú

prax, spoluprácu s ním na niekoľkých projektoch a takisto treba poznať znalosti jednotlivých členov tímu.

Pri počítaní LOC sa niekedy počítajú aj riadky binárnych súborov a dokumentácie. Tak sa prípadné väčšie množstvo administratívnej práce prejaví v počte riadkov. Ale považovať to za „jedinú správnu“ metódu nie je správne. Myslím, že manažér, ktorý sa o to pokúsi, časom zistí, že jeho najmenej produktívny človek tvrdo zapracuje na počte riadkov, ale napriek tomu zostane málo produktívny. K tomu vedie aj problém kvality oproti kvantite. Počet riadkov neurčuje koľko práce programátor spravil, hovorí iba koľko v ten deň napísal. Myslím, že dávať prednosť kvantite pred kvalitou pri produkte alebo službe je dosť chabý manažment. Dalo by sa to prirovnať k stavaniu múru z tehiel. Ak niekto postaví múr, ktorý sa mal stavať dva dni, za jeden deň, ale múr nie je taký rovný, ako ho chcel, stálo to za tie peniaze?

Ako počítať

Ak zoberieme do úvahy všetky spomenuté príklady počítania LOC v praxi, volá to po zavedení jednotného spôsobu počítania. Spôsobu, ktorý by bol najviac objektívny pre obe strany – programátora i vedenie. Táto úloha je však do dnes nevyriešená, pretože názory na to čo a ako počítať sa líšia.

Príkladov nájdeme v praxi hneď niekoľko. Už pri spomínaných komentároch je na jednej strane názor, že komentár nepridáva žiadnu novú funkcionalitu, preto by sa nemali počítať. Proti je takisto argument, že programátor môže komentáre „zneužívať“ na zbieranie ďalších riadkov, aby navonok preukazoval vyššiu produktivitu. Na druhej strane je názor, že komentár je potrebný pre lepšiu čitateľnosť a zrozumiteľnosť kódu. A keďže sa ním programátor musí zaoberať a venuje mu istý čas, malo by sa to odraziť aj vo veľkosti zdrojového kódu (počítaného podľa metódy LOC).

Podobne je to s testovacím kódom. Pri vyvíjaní softvéru programátor často napíše kód, ktorý slúži iba na umožnenie alebo aspoň uľahčenie testovania hotových častí projektu. Takýto kód sa píše „na zahodenie“, a vo výslednom produkte sa nikdy nevyskytuje. Znovu sú tu dve strany – jedna tvrdí, že čo nepridá žiadnu novú hodnotu ku koncovému výstupu, preto sa nepočíta. Druhá strana tvrdí, že testovací kód je potrebný na úspešné zvládnutie projektu. Je jeho neoddeliteľnou súčasťou, tak prečo ho vlastne nepočítať?

Ako posledný príklad uvediem ešte znovupoužitie kódu. Podľa jednej teórie sa počítajú iba riadky, kde sa na tento kód odkazuje (teda kde sa volá). Tu už veľmi neplatí dôvod, že programátor musí znovupoužitému kódu venovať „plnohodnotný“ čas. Ale existujú zástancovia názoru, že čas ušetrený na tomto mieste sa minie niekde inde (napr. pri podrobnejšom študovaní riešenej problematiky) a vytvára istú rovnováhu medzi počtom LOC a veľkosťou projektu, čo v konečnom dôsledku by malo zvýšiť presnosť odhadu pomocou tejto metódy. Ak si spomenieme na nápad s počítaním rozdielov (alebo zmien) v zdrojovom kóde, niekedy môže spôsobiť ťažkosti nezarátať znovupoužitý kód (ak by sme to mali aspoň automatizované, čo asi je pravdou vo väčšine prípadov).

LOC ako jeden z ukazovateľov

Ako z vidíme z vyššie opísaných príkladov, počítanie riadkov nie je jednoznačnou metódou, ktorá objektívne vyjadruje veľkosť projektu alebo meria produktivitu práce. Môžeme to tvrdiť už len preto, lebo neexistuje jednotný spôsob počítania, na ktorom by sa zhodli všetci, ktorí túto metódu používajú.

Jedna firma si spravila v tejto oblasti nasledovný experiment [2]. Zozbierala 20 rôznych potenciálnych ukazovateľov a sledovala aký vplyv budú mať na čas, potrebný na dokončenie projektu. Zistila, že iba 12 z týchto 20-tich ukazovateľov malo vôbec nejaký vzťah a iba 8 z nich malo vzťah, ktorý bol dostatočný na zváženie tohto ukazovateľa pri odhadovaní veľkosti projektu. Najvýznamnejším ukazovateľom sa stal počet zmien, s ktorými prišiel zákazník počas prvého mesiaca projektu. To očividne nemá veľa spoločného s počtom riadkov zdrojového kódu...

Armour dáva vo svojom príspevku [2] pekný príklad na porovnanie použitia počítania LOC s návštevou u lekára. Píše, že počítanie LOC a metóda funkčných bodov sú iba indikáciou nejakej zmeny, ale nič bližšie o nej nehovoria. Je to ako keď navštívime lekára s teplotou 38°C. Čo nám je? To nám lekár nebude vedieť povedať z tohto jediného ukazovateľa, ale nazbiera ďalšie ukazovatele a až keď ich bude mať „dostatočný“ počet, povie, že nazbierané ukazovatele indikujú nejaký konkrétny stav (napr. chorobu). Niekedy nebudú ukazovatele indikovať správne a musia byť vysvetlené. Podobne je to s LOC a ostatnými metódami. Ak sa od jedného systému očakáva, že bude mať 2-krát toľko riadkov ako druhý, indikuje to, že jeho vytvorenie asi potrvá oveľa dlhšie (zvyčajne proporcionálne k počtu LOC), ale nemusí to byť pravidlom.

V konečnom dôsledku robí odhad iba človek (väčšinou stanovený expert), ktorý pri zbieraní rôznych ukazovateľov môže dôjsť k iným výsledkom ako keby tie isté ukazovatele zbieral nejaký iný človek. Dvaja experti môžu dôjsť k úplne iným odhadom. Jeho posudzovanie je subjektívne vzhľadom na jeho znalosti a skúsenosti. Ak už robil viac odhadov, pravdepodobne spraví lepší odhad ako niekto, kto robí svoj prvý odhad. Niekedy sa termín určuje dokonca z druhej strany – od zákazníka. Zákazník si určí, že v tento termín to potrebuje a softvérová firma sa prispôsobí. Toto nie je najlepší prístup z hľadiska kvality, ale väčšinou sa termín dodrží, aj keď je niečo trochu ináč, ako sa plánovalo, alebo sú v systéme stále nejaké menšie chyby. Ale to je už iná téma.

Záver

V každom prípade počítanie LOC má zmysel. Nie ako „jediný správny“ ukazovateľ, ale ako jeden z množiny ukazovateľov. Odhad veľkosti projektu nemusí byť nutne správny (veď je to iba odhad). Naznačí nám, koľko by trvalo realizovanie priemerného projektu v oblasti, ktorú riešime. Máme aspoň približnú predstavu o tom, koľko to *môže* trvať a môžeme sa pokúsiť oprieť o iné metódy, aby sme sa priblížili skutočnej hodnote. Podobne je to pri meraní produktivity práce. LOC nám indikuje pokles alebo

vzrast produktivity, ale je na nás zistiť prečo vznikli tieto výkyvy a posúdiť, či sú relevantné.

Použitá literatúra

1. Albrecht, A.J.: Measuring application development productivity. In: *Proceedings of the IBM Application Development Symposium*, Monterey, CA (Oct. 1979) 83-92.
2. Armour, P.G.: The business of software: Beware of Counting LOC. *Communications of the ACM,* Vol. 47, No. 3 (2004) 21-24
3. Armour, P.G.: The business of software: Ten Unmyths of Project Estimation. *Communications of the ACM,* Vol. 45, No. 11 (2002) 15-18
4. Jones, C.T.: Backfiring: Converting lines of code to function points. *IEEE Computer 28,* Vol. 11 (Nov. 1995)

Annotation

Counting lines of code

In the present the method of counting lines of code is commonly used for time estimation (usually in the man-month unit) needed for a successful project completion. Also it is used for work production measurement of a programmer or a team. In this paper I describe the various ways to look at the counting of lines of code method. I focus mainly on the problems which arise by using this method the two areas I mentioned before. I explain in more detail the various approaches of how to count the lines of code and present a draft of alternative possibilities of measuring productivity. Each method has some disadvantages. That is why my goal in this paper is to describe a way to use this method considering its disadvantages.