PREČO UŽ NEODHADOVAŤ ROZSAH PROJEKTU POČTOM RIADKOV KÓDU

Real Programmers don't comment their code. If it was hard to write, it should be hard to read.

Michal Námešný

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
misinko[zavináč]gmail[.]com

Abstrakt. *Správny odhad softvérového projektu je dobrým predpokladom pre jeho úspešné ukončenie, najmä ak sa jedná o rozsiahlejší projekt. Základom je správne odhadnúť jeho rozmer, teda jeho veľkosť, od ktorej sa odvíja množstvo vynaložených ľudských zdrojov potrebných pre jeho zvládnutie, celkový čas potrebný na jeho dokončenie a v konečnom dôsledku jeho cena. Na určenie rozsahu projektu sa vymysleli rôzne metriky, pomocou ktorých by sme ho dokázali exaktne zmerať a mohli navzájom rôzne projekty porovnávať. V eseji zhodnocujem, či má stále zmysel odhadovať rozsah projektu dlhoročne používanou metrikou - počet riadkov kódu a či by nebolo vhodnejšie zamerať sa na jej alternatívy. Rozoberám nedostatky odhadu pomocou spomenutej metriky a prečo už dnes nemusí byť taká spoľahlivá ako v minulosti.*

Kľúčové slová: *odhad softvérového projektu,odhad rozsahu, počet riadkov kódu, LOC*

Úvod

V praxi sa často nestáva, že by zákazník dostal po ukončení projektu výsledný produkt lacnejšie, ako bolo pôvodne naplánované. Alebo, aby sa projekt úspešne zavŕšil s veľkým časovým predstihom. Skôr môžeme byť svedkami neúspešných projektov, ak zákazník nemal záujem o navýšenie rozpočtu po ich prvotnom, zlom cenovom odhade. Pri zlom

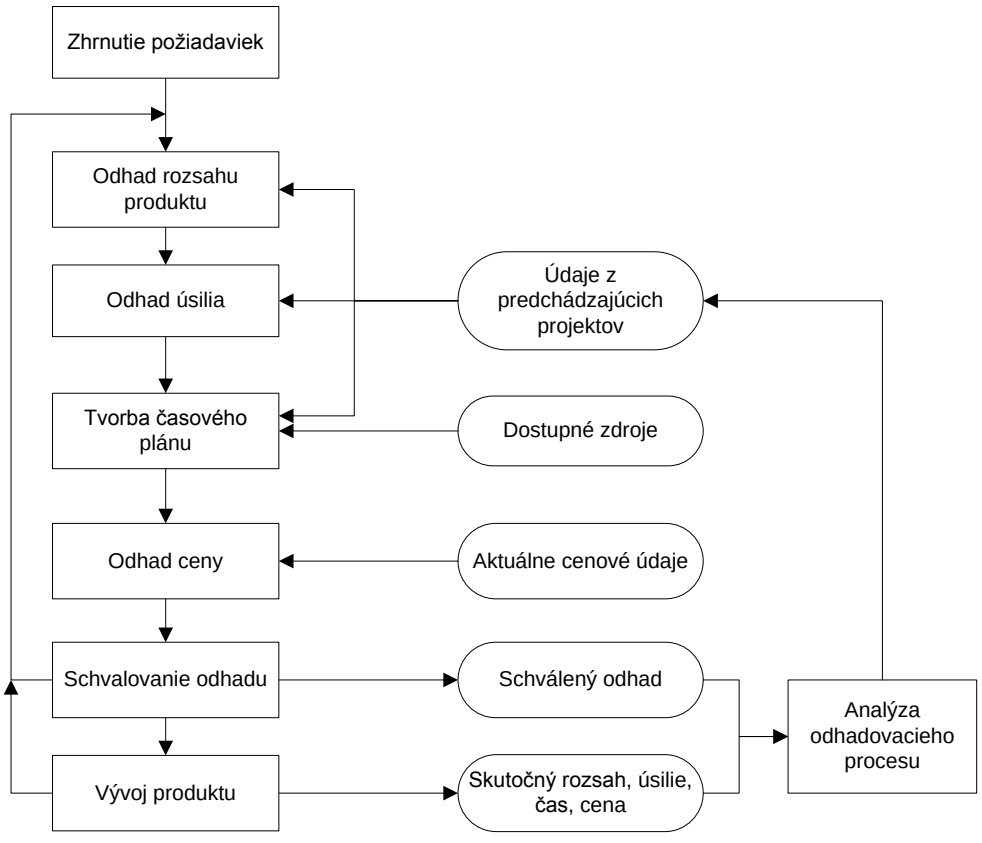
časovom odhade môže byť spoločnosť dokonca sankcionovaná pre nedodržanie termínu. Správny odhad preto môže veľmi pomôcť projektu pre jeho bezkonfliktný priebeh. Na to, ako ho zistiť, existuje veľa metód. Avšak v akých jednotkách ho definovať?

Odhad softvérového projektu

Podľa [2] odhad softvérového projektu pozostáva zo štyroch základných krokov:

1. Odhad rozsahu projektu
2. Odhad úsilia v človeko-mesiacoch, alebo v človeko-hodinách použitím modelu zohľadňujúci rozsah projektu a produktivitu tímu
3. Vytvorenie časového plánu podľa kalendárnych mesiacov
4. Odhad ceny – kombinácia prostriedkov ktoré je potrebných vynaložiť na softvérovú a hardvérovú časť projektu

Základný odhadovací proces je znázornený na Obr. 1, kde môžeme vidieť, že kľúčovým krokom je odhadnutie rozsahu projektu, od ktorého sa odvíjajú zvyšné odhady. Nesprávny odhad rozsahu môže znamenať aj neúspech celého projektu. V ďalších častiach eseje sa práve preto zameriam na prvý krok odhadovacieho procesu.



Obr. 1. Odhadovací proces projektu [2].

Odhad rozsahu softvérového projektu

Zmerať rozsah softvérového projektu nie je také jednoduché ako odmerať dĺžku stola, alebo odstopovať čas potrebný na zabehnutie sto metrov. Odhadnúť ho, je o to zložitejšie. Prečo je to také zložité môžeme vidieť v Tab. 1. Softvér oproti klasickým produktom častokrát vychádza zo slabej špecifikácie a návrhu, ktorá je postupne dodefinovaná počas jeho vývoja. Je ťažké niečo zmerať, keď ani nevieme aké to bude nakoniec veľké. Istú dávku nevedomosti môžu vnášať aj zmeny, ktoré je potrebné vykonať pri vyskytnutí sa problémov, na ktoré budeme musieť dynamicky reagovať, hlavne ak riešime projekt v doteraz nám neznámej oblasti.

Tab. 1. Porovnanie vlastností typického produktu a softvéru [3].

#	Vlastnosť	Typický produkt	Softvér
1	funkcie	definitívne, presne definované	môžu byť nejasné, môžu sa zmeniť počas vývinu
2	kvalitatívne vlastnosti	presne merateľné	merateľné, závislé od skúseností členov tímu
3	dostupnosť nástrojov	nástroje dostupné na trhu	nástroje dostupné
4	potrebné zručnosti a vedomosti pracovníkov	známe	známe
5	potrebné úsilie	môže byť odhadnuté veľmi presne	voľný odhad
6	odhad ceny	jednoznačný	nejednoznačný odhad manažéra
7	zmeny počas produkcie	zanedbateľné	veľmi časté
8	špecifikácia a návrh	presne definované, odsúhlasené pred začatím produkcie	nejasne definované, zmeny počas celého vývojového cyklu
9	prepracovanie, vylepšenie	nemožné po doručení	stále možné, aj po doručení

Na odhad rozsahu projektu existuje mnoho metód a spôsobov. My si predstavíme dva základné.

1. *Analogicky* – Najpriamejší spôsob odhadnutia rozsahu projektu. Predpokladom je, že sme už v minulosti riešili podobné projekty, alebo aspoň jeho časti a máme informácie o ich veľkostiach. Týmto spôsobom dokážeme získať celkom presný odhad ak, ako sme už spomínali, oblasť nového projektu je nám dobre známa. Presnosť odhadu môže, teda nepochybne aj bude narastať so skúsenosťami manažéra, ktorý odhad vytvára.
2. *Analyticky* – Rozdelením softvéru na malé jednotivé časti a spočítaním funkcií ktoré má obsahovať podľa špecifikácie. A potom aplikovaním algoritmického prístupu vypočítame počet funkčných bodov, alebo prípadov použitia.

Avšak oveľa väčším problémom ako zmerať rozsah je, v akých jednotkách ho zmerať. Nemáme totiž k dispozícii veličinu, ktorá by nám dala jednoznačnú informáciu. Veličinu pod ktorou by si každí exaktne predstavil rovnaký rozsah softvéru. Najčastejšie sa rozsah

uvádza v metrikách počet riadkov kódu, alebo vo funkčných bodoch. Sú to dve z najpoužívanejších metrík, pričom každá má svoje pozitívne aj negatívne vlastnosti.

Metriky pre rozsah softvéru

Počet riadkov kódu

Počet riadkov kódu je jedna z najstarších a najpopulárnejších metrík. Je ešte stále vhodné ju používať? V tejto kapitole poukážem na jej nedostatky. Nedostatky najmä také, ktoré sa objavili príchodom objektovo-orientovanej paradigmi a moderných integrovaných vývojových prostredí.

Základnou nevýhodou tejto metriky bolo, že nemala všeobecne štandardizované pravidlá. Každý si ju vysvetľoval svojvoľne, a preto počet riadkov kódu podobných projektov v dvoch rôznych spoločnostiach, mohol byť výrazne odlišný. V posledným pár rokoch sa urobil v tomto smere výraznejší pokrok, aj keď o jednotnej štandardizácií nemôžeme hovoriť. Odhadovať rozsah analogicky preto môžeme len z projektov, na ktorých sme spočítali riadky rovnakým algoritmom. Riadky sú samozrejme počítané programom, ale musíme mať k dispozícií ich zdrojové kódy.

```
// Toto je jeden riadok kódu.
for (int i=0; i<n; i++) sum += i;

// A toto sú štyri?
for (int i=0; i<n; i++)
{
  sum += i;
}
```

Výsledný počet zdrojových riadkov kódu sa dozvieme v momente keď už ho nepotrebujeme, teda pri ukončení projektu. Môže nám poslúžiť už len v budúcnosti ak sa budeme venovať podobnému projektu ako údaj pre ďalší odhad. Avšak špecifikom softvéru je, že je ho potrebné neustále dolaďovať a upravovať podľa požiadaviek zákazníka, a tak ani finálny počet riadkov, nemusí byť vôbec konečný.

Počet riadkov ako taký nám úvádza koľko textu museli programátori napísať. Vôbec neodzrkadľuje prácu analytikov, dizajnérov, testerov, či manažérov. Pri ďalších krokoch odhadu teda nesmieme zabudnúť na to, že každý riadok kódu so sebou nesie aj prácu ostatných členov tímu.

Ďalším podstatným nedostatkom tejto metriky je závislosť od programovacieho jazyka. Určite bude každý jeden so mnou súhlasiť, že program v jazyku C# bude mať podstatne menej riadkov ako v jazyku C. Aj keď existujú prevodové tabuľky, musíme poznať pravidlá počítania riadkov kódu v oboch jazykoch, podľa ktorých boli tabuľky vytvorené.

Tu už sa dostávame k jadru problému. V Tab. 2 môžeme vidieť prvých desať najobľúbenejších programovacích jazykov za mesiac október 2009.

Tab. 2. Obľúbenosť programovací jazykov [4].

Pozícia Október 2009	Programovací jazyk
1	Java
2	C
3	PHP
4	C++
5	(Visual) Basic
6	C#
7	Python
8	Perl
9	JavaScript
10	Ruby

Ako poukázal autor [1], pre väčšinu z týchto jazykov existuje integrované vývojové prostredie, ktoré poskytuje nástroje na jednoduché vytváranie grafických používateľských rozhraní, automatickú generáciu konštruktorov tried, hlavičiek metód, dokonca aj celých metód s minimálnymi požadovanými úpravami pre programátora. Myslím si že použitím týchto vývojových prostredí stráca metrika počet riadkov kódu ako taký zmysel, pretože množstvo kódu sa automaticky generuje na pozadí. Je teda takpovediac nemožné spočítať riadky, ktoré sa len vygenerujú a na ktorých bude musieť programátor reálne pracovať. Keď sa pozrieme opäť do tabuľky, vidíme že väčšinu jazykov môžeme zaradiť do skupiny objektovo-orientovaných. Táto paradigma priniesla so sebou množstvo otázok a aj také jednoduché zapuzdrenie môže byť problémom pre správne počítanie riadkov.

```
// Zapuzdrenie v jazyku C#, 4 riadky, so zátvorkami 6 riadkov
private int a;

public int A
{
  get { return a; }
  set { a = value; }
}

// Alebo nechať premennú s právami public? 1 riadok.
// Zvyčajne nevhodné, ale niekedy to môže byť užitočné.
public int a;
```

Ak sa pozrieme do štatistiky [4] na procedurálne jazyky, pri ktorých bolo spočítanie zdrojových riadkov omnoho jednoduchšie, nájdeme Fortran na 26. pozícií a Adu na 29. pozícií.

S problémom počítania generovaného kódu sa môžeme stretnúť aj pri použití techniky modelom riadeného vývoja softvéru. Aj keď sme museli vynaložiť určité úsilie na návrh modelu, máme započítavať aj riadky ktoré boli strojovo vygenerované? Na druhej

strane, návrh a úprava modelu nám zjednodušia prácu ako by sme mali softvér preprogramovať.

Funkčné body

Funkčné body sú akousi funkcionálnou jednotkou pre meranie rozsahu softvéru. Vychádza zo špecifikácie a určuje počet funkcií, alebo služieb ktoré má poskytovať používateľovi. V čom sú lepšie oproti počtu riadkov kódu? A kedy ich zas nie je vhodné použiť.

Jednou z najväčších výhod oproti metrike počtu riadkov kódu je práve fakt, že vychádza zo špecifikácie, a teda je táto metrika nezávislá na implementácií. Vďaka tomu môžeme projekty navzájom lepšie porovnávať. Z druhého uhľa pohľadu sa však môže vyskytnúť problém, ak nemáme špecifikáciu kompletnú. V tomto prípade však pravdepodobne zlyhá väčšina metód, keďže presne nevieme, čo konkrétne zákazník očakáva.

Ďalšou pozitívnou vlastnosťou je, že ak dáme dvom skúseným manažérom odhadnúť rovnaký projekt, výsledný odhad by im mal vyjsť približne rovnaký. Je samozrejmé, že im pravdepodobne nevyjde to isté číslo, keďže vnášajú do procesu istú dávku subjektivity, ale rozdiel by nemal byť príliš veľký.

Použitie funkčných bodov si však vyžaduje čas, pretože ich nie je možné tak jednoducho a rýchlo spočítať ako počet riadkov kódu a istú dávku zručností a skúseností manažéra. Napriek tomu si myslím, že každý manažér by si mal osvojiť nejakú techniku na výpočet funkčných bodov.

Alternatívne metriky

Prípady použitia

Výhodou tejto metriky je nezávislosť na implementácii, pretože vychádza priamo z analýzy, konkrétne z modelu prípadov použitia jazyka UML2, ktorý podporuje objektovo-orientovanú paradigmu vývoja softvéru. Analýza sa vytvára ešte v prvých etapách projektu, vďaka čomu je táto metrika vhodná na prvotný odhad softvéru. Ak však manažér chce konkrétnejší odhad, či časový plán, prípady použitia už nemožno použiť, pretože sú na vyššej úroveni abstrakcie ako je potrebné.

Jej silnou stránkou je, že pomocou UML2 dokáže rozložené časti systému graficky ilustrovať a dať aj zákazníkovi základnú predstavu o veľkosti požadovaného systému.

Body vlastností

Boli vytvorené kvôli nedokonalosti funkčných bodov pri odhadovaní softvéru s veľkou algoritmickou zložitosťou ako sú obranný raketový systém, operačný systém, radarový navigačný systém, matematické výpočtové prostredia a podobne.

Vychádzajú z funkčných bodov, pričom len upravujú a dodefinuvávajú ich parametre. Ak spočítame funkčné body a body vlastností menšieho informačného systému, ich hodnota nám vyjde pravdepodobne rovnaká. Avšak hodnota bodov vlastností zložitejšieho softvéru je podstatne vyššia ako hodnota funkčných bodov.

Záver

Aj napriek všetkým nevýhodám metriky počet riadkov kódu, si nemyslím že by prestala mať svoj význam. Teda ako základ pre odhad rozsahu softvéru už podľa mňa nie je spoľahlivá, ale ak sa presne definujú všeobecné pravidlá pre počítanie riadkov pre jednotlivé jazyky, mohol by to byť ďalší údaj ktorý by nám poskytoval nejakú informáciu o projekte. Nebol by samozrejme vždy presný, ale čím viac informácií o projekte vieme, tým máme väčšiu šancu správne naplánovať jeho priebeh. Perspektívne do budúcnosti sú podľa mňa prípady použitia, ktorými dokážeme veľmi prehľadne a zrozumiteľne znázorniť systém aj zákazníkovi, pretože aj on si musí uvedomiť rozsah projektu, ktorý od spoločnosti žiada. Preto práve prípady použitia v kombinácií s funkčnými bodmi, by mohli byť základnými nástrojmi pre prácu manažéra.

Použitá literatúra

1. Galorath, D.: Have We Lost Our Ability to Estimate Software Size?. Dostupné na internete: http://www.galorath.com/wp/have-we-lost-our-ability-to-estimate-software-size-2.php, [cit: 2009-Október]
2. Hewson, G., Peters, K.: *Fundamentals of Software Project Estimation.* Software Productivity Center Inc., 2007.
3. Parthasarathy, M.A.: *Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects.* Addison-Wesley, 2007.
4. Tiobe software: *TIOBE Programming Community Index for October 2009.* Dostupné na internete: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, [cit: 2009-Október]

Annotation

Why don't estimate size of project in lines of code

Right estimate of the software project is good assume for its successful completion, especially if there is an extensive project. It is essential to estimate the correct size, on which depends the amount of needed human resources, the total time needed for completion and ultimately its cost. There are different metrics to estimate the size of project, by which we would be able to exact measure and compare different project to each other. The essay deal with whether it still makes sense to estimate the size of the project for many years used metric - lines of code, and whether it would be preferable to focus on its alternatives. I discuss the shortcomings mentioned estimation using metrics and why now may not be so reliable as in the past.