Odhad a naplánovanie práce na školskom softvérovom projekte

MIROSLAV KALLO

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
kallo02@student.fiit.stuba.sk

Abstrakt. Hádam každý softvérový inžinier začínal svoju činnú kariéru prácou na rôznych školských projektoch, či už samostatných alebo tímových. Hoci samostatná práca býva obvykle ľahšia, pretože nie je nutné koordinovať činnosť viacerých členov tímu, v oboch prípadoch je dôležitý správny manažment práce a plánovanie. Treba správne a hlavne včas odhadnúť daný projekt, aby bolo možné vytvoriť čo najpresnejší plán projektu, pretože neskoré odovzdanie znamená v škole väčšinou totálny neúspech. V tejto eseji sa budem zaoberať problémami plánovania projektov v škole a predostriem základnú sadu rád a ponaučení, pomocou ktorých šanca úspešne absolvovať softvérový projekt značne narastie.

Úvod

Napriek tomu, že za celú dobu vývoja informačných technológií sa veľa odborníkov zaoberalo problematikou plánovania a manažmentu, vývojári ich stále podceňujú. Týka sa to hlavne menších projektov, ktoré sa zdajú byť príliš jednoduché na to, aby sa podrobne naplánovali. Nikto si však neuvedomuje, že každý človek si intuitívne vytvára plány a rozvrhy - plánuje si každodenné činnosti. Takéto plány sú ale príliš všeobecné a povrchné na to, aby sa podľa nich dal úspešne uskutočniť softvérový projekt.

Podľa [1] dobrý projektový plán:

- *redukuje neurčitosť (výsledku projektu)*
- *zlepšuje výkonnosť*
- *zabezpečuje lepšie porozumenie cieľom projektu*
- *vytvára základňu pre sledovanie a riadenie práce*

Manažment v softvérovom inžinierstve, október 2006, s. 1-9.

Na vytvorenie plánu je potrebné najskôr odhadnúť veľkosť a zložitosť projektu (viď Obr. 1). V praxi sa však presnému odhadu softvérového projektu venuje neadekvátne malé množstvo úsilia, čo má za následok rôzne problémy.

Podľa [3] napríklad štúdia vlády Spojených štátov amerických odhalila, že:

- 60% projektov sa oneskorilo
- 50% projektov bolo predražených
- 45% doručených projektov nebolo použiteľných

Situácia školských softvérových projektov je o to ťažšia, že študenti majú malé skúsenosti s vývojom a manažmentom (nielen softvérových) projektov. Navyše sa v tom istom čase podieľajú na realizácii viacerých projektov. Preto správny rozvrh času je tu o to dôležitejší.

Plánovanie softvérového projektu

Podľa [1] je *projekt* „dočasné úsilie, ktoré sa vyvíja s cieľom vytvorenia jedinečného výrobku alebo služby“. Projekt je teda proces. Má definovaný svoj začiatok aj koniec a je kombináciou tvorivých procedúr s úsilím dosiahnuť cieľ.

Plánovanie softvérového projektu v sebe zahŕňa viaceré procesy. Tieto sa buď priamo podieľajú na tvorbe rozvrhu a rozpočtu, alebo dopĺňajú základné procesy o aspekty z ostatných oblastí manažmentu softvérového projektu. Medzi *základné procesy plánovania* patria podľa [1]: *manažment rozsahu*, *manažment rozvrhu*, *manažment nákladov* a *manažment integrácie plánu projektu*.

Jednotlivé procesy zahŕňajú v sebe tieto činnosti:

- *Manažment rozsahu* - tu sa stanovuje rozsah projektu a jednotlivých komponentov
- *Manažment rozvrhu* - časť, kde sa definujú činnosti, ktoré sa po odhadnutí zoradia a vytvoria rozvrh
- *Manažment nákladov* - sem patrí manažment zdrojov a odhadovanie nákladov
- *Manažment integrácie plánu projektu* - posledná fáza, ktorá integruje výsledky ostatných procesov do jedného konzistentného dokumentu

Počas trvania projektu je naša pozornosť odvádzaná na rôzne iné detaily, ktoré si vyžadujú našu pozornosť. Tieto sa nazývajú *narušenia* (angl. disturbances). Základnú klasifikáciu *narušení* je možné vidieť v Tab. 1. Tieto *narušenia* predstavujú riziko pre projekt, a preto je treba ich v pláne zohľadniť. Ich analýzou sa bližšie zaoberá *manažment rizík*, ktorého výsledky sú potom použité pri tvorbe rozvrhu.

Typy narušení	Silný vplyv	Slabý vplyv
externé	x	
interné		x
priamo týkajúce sa projektu	x	
priamo týkajúce sa úlohy		x
nepriamo týkajúce sa projektu	x	
nepriamo týkajúce sa úlohy	x	x
okamžité		x
dlhodobé	x	
objektívne		x
subjektívne	x	

Tab. 1 Klasifikácia narušení [4].

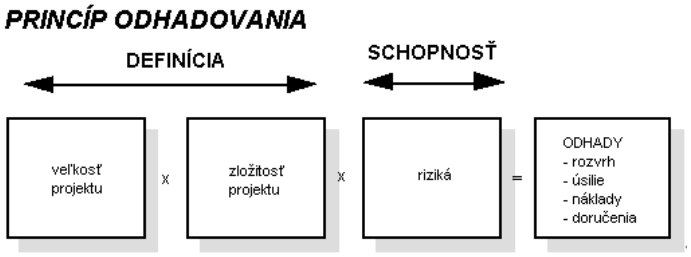
Medzi ďalšie aspekty, ktoré v závislosti od typu projektu treba brať do úvahy pri plánovaní [1], patria *manažment kvality*, *manažment ľudských zdrojov*, *manažment obstarávania* a *manažment komunikácie*.

V školských projektoch je postačujúce, ak sa týmto procesom venujeme pomenej. Napríklad v *manažmente ľudských zdrojov* je postačujúce, ak si rozdelíme roly v tíme a nebudeme sa zaoberať získavaním personálu, pretože počet členov v tíme býva pevne stanovený a je nemenný. Taktiež nie je nutné zaoberať sa preberacími protokolmi pri obstarávaní externých zdrojov, pretože ak sa pri vývoji majú také zdroje použiť, býva podmienkou, aby boli verejne prístupné a voľne šíriteľné. Riziká nemusíme úplne vyčísľovať, ale určite si ich musíme byť vedomí, aby sme im mohli predchádzať alebo aspoň zmierňovať ich účinky.

Mali by sme sa zaoberať hlavne základnými procesmi: mali by sme si vytýčiť ciele a určiť rozsah vypracovania projektu, určiť činnosti a zostaviť rozvrh projektu. Rozpočet v školských projektoch zahŕňa obvykle silné ohraničenie času, ktorý môžu študenti stráviť prácou, dostupnosťou hardvéru a softvéru na školách a peniazmi – pokiaľ projekt nie je sponzorovaný firmou, mal by byť zadarmo. Nesmieme zabúdať ani na projektovú dokumentáciu – patrí sem aj plán projektu, preto je dôležitá jeho integrácia do konzistentnej dokumentácie.

Odhadovanie softvérového projektu

Prvým krokom v plánovaní softvérového projektu je naplánovanie a definovanie rozsahu, aby bolo možné odhadnúť náklady, čas a úsilie, ktoré bude potrebné vynaložiť na jeho uskutočnenie. Všeobecná schéma princípu odhadovania softvérového projektu je znázornená na nasledujúcom Obr. 1.



Obr. 1 Princíp odhadovania [3].

Celkový odhad závisí od definície problémovej domény a od našej schopnosti daný projekt uskutočniť. Problémovú doménu môžeme jednoducho opísať rozsahom projektu, ktorý závisí od jeho veľkosti a zložitosti. Naša schopnosť zrealizovať daný projekt zase závisí od rizík, ktoré môžu vplývať na samotný proces realizácie.

Na vytvorenie odhadu je možné použiť niektorú zo známych techník odhadovania (dekompozícia, počítanie dĺžky programu napr. LOC (Lines of Code), funkčné body (angl. function points), body vlastností (angl. feature points), 3D funkčné body (angl. 3D function points), metóda Mark II, model COCOMO (COnstructive COst MOdel), body prípadov použitia (angl. use case points), a iné). Jednotlivé techniky sa líšia zložitosťou výpočtu, presnosťou výsledného odhadu, množstvom potrebných vstupných údajov a možnosťou skorého nasadenia.

V nasledujúcich riadkoch budú popísané bližšie niektoré najčastejšie používané techniky.

Dĺžka textu programu

Je to najjednoduchšia metrika. Na určenie odhadu sa používa hlavne analógia s inými projektmi, pretože skoro vždy máme k dispozícii historické údaje (zdrojové texty programov).

Najčastejšie sa používajú tieto spôsoby vyjadrenia dĺžky programov [1]:

- počet oddeľovačov
- počet dodaných príkazov (angl. DSI - delivered source instructions)
- počet riadkov zdrojového textu (bez komentárov a prázdnych riadkov)
- počet znakov
- počet bajtov potrebných na uchovanie zdrojových textov
- počet strán výpisu programu

Hlavné výhody tejto techniky sú jej *jednoduchosť* a široká použiteľnosť. Naopak, táto technika *poskytuje veľmi nepresné odhady* na začiatku projektu. Presnejšie odhady

ponúka až v neskorších fázach projektu, kedy máme k dispozícii časti zdrojových textov programu.

Funkčné body

Táto technika vychádza zo špecifikácie požiadaviek a určuje sa tu počet a rozsah používateľovi prístupných služieb systému.

Pri analýze požiadaviek sa určujú nasledujúce charakteristiky [1]:

1. *počet externých vstupov*
2. *počet externých výstupov*
3. *počet logických interných súborov*
4. *počet externých rozhraní*
5. *počet externých dopytov po údajoch*

Dané charakteristiky sa potom ohodnotia *stupňom zložitosti* jednoduchý, stredný alebo zložitý. Potom sa vypočítajú hodnoty funkčných bodov. Viac informácii o výpočte a presné vzťahy môže čitateľ nájsť v [1].

Garmms a Herron v [3] označujú funkčné body za techniku, ktorá poskytuje najlepšiu *presnosť* a *flexibilitu* odhadovania. Je pravdou, že je *možné použiť* ju *v skorých etapách projektu* a je *nezávislá od použitého programovacieho jazyka*, na druhej strane autor sám poukazuje na jej nedostatky:

Pokiaľ nemáme úplnú špecifikáciu, *celkový poskytovaný odhad* je nepresný a *zodpovedá len úrovni špecifikovaných vlastností projektu*. Proces sa *nedá automatizovať* a je tu *vysoký stupeň subjektivity* pri určovaní charakteristík [1].

Navyše podľa [2] na určenie funkčných bodov sú *potrební experti* v danej oblasti, čo v prípade školských softvérových projektov znamená určitý problém, pretože účastníci projektu sú študenti s malými praktickými skúsenosťami a nemajú ešte osvojené „najlepšie praktiky“ pri práci.

3D funkčné body

Jedná sa o rozšírenie techniky funkčných bodov. Základom je myšlienka, že problém sa dá vyjadriť v troch rozmeroch, ktorými sú *údaje*, *funkcie* a *riadenie*. Každý rozmer má svoje charakteristiky, ktoré spolu tvoria celkovú zložitosť problému. Niekedy je jeden rozmer dominantný, ale pre správne určenie odhadu je potrebné vyčísliť všetky tri rozmery.

Rozmer údajov je typický pre databázy a informačné systémy. Jeho charakteristiky sú odvodené priamo od charakteristík funkčných bodov. *Funkčne silné* problémy sú typické pre vedecký sektor. Charakteristiky funkčného rozmeru obsahujú počet a zložitosť použitých funkcií. Nakoniec, k problémom vyžadujúcim *silné riadenie* patria systémy reálneho času. Sú charakterizované stavmi systému a prechodmi medzi týmito stavmi.

COCOMO

Prvá verzia COnstructive COst MOdel bola predstavená Barry Boehmom v roku 1981. Model definuje potrebné úsilie a čas pre projekt. Vychádza sa z odhadu veľkosti projektu.

Má tri úrovne [1]:

- *základná* (len hrubý odhad)
- *stredná* (berie do úvahy aj atribúty systému)
- *podrobná* (rozdeľuje úsilie medzi etapy projektu)

V každej úrovni je zložitosť systému vyjadrená pomocou jedného z troch módov, ktoré sú: *organický*, *prechodný* a *viazaný*. Viac informácii o výpočte a presné vzťahy môže čitateľ nájsť v [1].

Počas vývoja vzniklo viacero modifikácií modelu COCOMO. Verzia 2 používa ako základ na odhadnutie veľkosti v prvej etape projektu techniku *objektových bodov*, v druhej etape používa *funkčné body* a až v tretej etape používa *počítanie riadkov kódu* (LOC), čo jej zaručuje vyššiu mieru flexibility. Ako môžeme ale vidieť, aj táto technika *neposkytuje* veľmi *presné odhady v prvej etape* projektu.

Use case points

Táto technika tak ako aj ostatné používa na určenie odhadu úsilia ako vstupný parameter veľkosť projektu. V objektovo-orientovanej analýze a návrhu sa na opísanie funkčných požiadaviek používa model prípadov použitia (angl. Use case model). Z tohto modelu je preto možné už v prvej etape projektu odhadovať jeho veľkosť. Aj pre skorý odhad je však potrebné mať aspoň čiastočnú znalosť problémovej oblasti, taktiež treba vedieť niečo o veľkosti a architektúre vytváraného systému.

V odhadovaní softvérových projektov pomocou metódy prípadov použitia (Use Case Points, UCP) sa vychádza z analýzy vytvoreného modelu prípadov použitia.

Klasifikácia hráčov a prípadov použitia

Počet hráčov a prípadov použitia je zrejmý z analýzy modelu prípadov použitia. Jednotlivých hráčov a prípady použitia je potrebné ešte klasifikovať. Pre obe skupiny sa používajú rovnaké triedy zložitosti: *jednoduchá*, *stredná* a *zložitá*.

- *Jednoduchým hráčom* je napríklad iný systém, ktorý s daným systémom komunikuje cez presne definované rozhranie.
- *Stredne zložitým hráčom* je iný systém, ktorý komunikuje pomocou protokolu.
- *Zložitým hráčom* je človek.

Na zaradenie prípadov použitia existuje viacero metrík, napr. počet transakcií alebo počet implementujúcich tried pre daný prípad použitia.

Celkovú neupravenú váhu pre hráčov (UAW) a prípady použitia (UUCW) vypočítame ako súčet počtov jednotlivých typov hráčov (prípadov použitia) vynásobených príslušnou váhou.

Technické vplyvy a vplyvy prostredia

Vypočítané neupravené váhy sa potom upravujú podľa rôznych vplyvov. Zoznam vplyvov môže čitateľ nájsť v [2]. Technické vplyvy zodpovedajú príslušným charakteristikám určených metódou funkčných bodov. Vplyvy prostredia sa používajú na opis nefunkcionálnych závislostí, napr. motivácia alebo jednoduchosť použitia.

Ako to celé vyzerá v školských projektoch?

V úvode dokumentu som vyzdvihol potrebu plánovania školských projektov. Skutočne, potreba plánovania je tu dôležitá kvôli špecifickej povahe takýchto projektov:

1. *účastníkom je jednotlivec alebo malý tím* (cca. 5 ľudí)
2. *účastníci sa podieľajú v tom istom čase na realizácii viacerých projektov*
3. *motiváciou nie je finančné ohodnotenie*
4. *jedná sa o projekty menšieho rozsahu*
5. *koniec projektu je striktne definovaný*
6. *rozpočet je nulový*

Plánovanie je dôležité hlavne kvôli bodom 2 a 5. Akú metódu si ale zvoliť, aby sme včas odhadli úsilie, ktoré bude potrebné vynaložiť, a mohli si vytvoriť rozvrh? So spomínaných metód sa najlepšou javí *metóda bodov prípadov použitia* (use case points).

Dnes je už objektovo-orientovaná analýza a návrh celkom bežná. Na zobrazenie sa používajú napr. UML diagramy a jedným z nich je aj *diagram modelu prípadov použitia*. Tento diagram dobre popisuje funkcionálne požiadavky a za predpokladu, že bol definovaný úplne, t.j. špecifikácia požiadaviek bola kompletná, predstavuje podrobný zdroj informácií pre odhad veľkosti a zložitosti systému hneď v prvej etape. Metóda má jednoduché kritériá ohodnocovania, a preto na jej vypracovanie nie sú potrební experti. Poskytuje aj pomerne presné odhady.

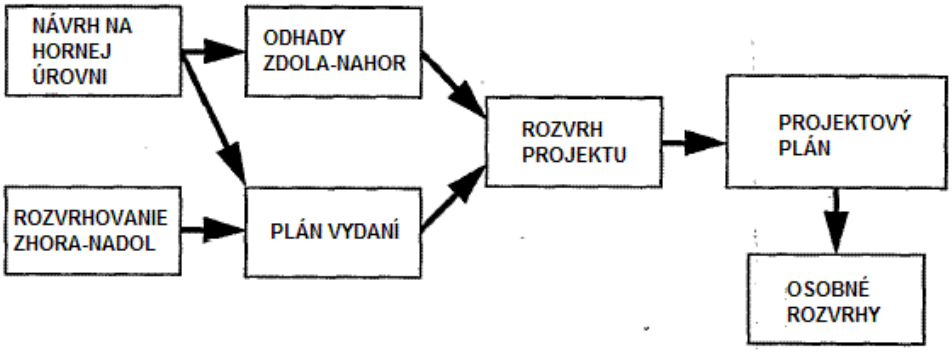
Možnou alternatívou je metóda, ktorá sa sústreďuje okolo architektúry systému. Je to vlastne alternatívny pohľad na systém, kedy sa berú do úvahy súčiastky systému, ktoré v sebe nesú jeden alebo viacero prípadov použitia. Daniel J. Paulish opísal v [5] svoje skúsenosti s takýmto prístupom.

Celá metóda je založená na tom, že keď máme navrhnutú architektúru systému, môžeme veľmi presne odhadnúť úsilie, ktoré treba vynaložiť na realizáciu projektu. Problémom samozrejme môže byť práve správne a dosť podrobné definovanie architektúry. Vzhľadom na existenciu štandardných modelov architektúr je možné celkom ľahko vytvoriť architektúru nášho systému, ak použijeme analógiu.

V súčasnosti sa podieľam na tímovom projekte, ktorého cieľom je vytvoriť hru. Každá hra má nejaké základné súčiastky (grafický engine, používateľské rozhranie,

jadro, ovládanie a iné), preto vytvoriť prvý návrh architektúry bolo veľmi ľahké. Ďalej bolo potrebné postupovať zhora-nadol a špecifikovať jednotlivé moduly.

Paulish v [5] uvádza postup, ktorým jeho vývojový tím postupoval pri riešení. Použili pri tom rôzne techniky odhadu (LOC, COCOMO), ktorých výsledky spriemerovali. Začali s architektonickým návrhom na hornej úrovni a paralelne vytvárali rozvrh systémom zhora-nadol. Potom zo získaných údajov vytvorili plán vydaní a vykonali odhad zdola-nahor pre všetky súčiastky. Tieto dva odhady potom slúžili na vytvorenie výsledného rozvrhu pre projekt. Celá procedúra je na Obr. 2.



Obr. 2 Plánovanie sústredné na architektúru [5].

Aj keď daný projekt nebol malého rozsahu, daný princíp sa pre naše podmienky použiť dá. Na určenie odhadu je vhodné zvoliť si metódu, ktorá nám vyhovuje viac, pretože obe sú približne rovnako dobré. Ak nemáme dobre definovanú architektúru, musíme postupovať podľa identifikovanej funkcionality, ktorá je zachytená v modeli prípadov použitia. Je však veľká šanca, že môžeme použiť analógiu pri výbere architektúry. Potom je dôležité použiť aj prístup zhora-nadol aj zdola-nahor, aby výsledky boli čo najpresnejšie.

Záver

Ani v školských projektoch nie je vhodné podceňovať plánovanie. Odhady veľkosti a potrebného úsilia musíme mať v skorých etapách projektu, preto je dôležité vybrať si správnu metódu na ich získanie. Zároveň metóda musí byť dostatočne jednoduchá, aby sa dala použiť v školských podmienkach.

Na riešenie takejto situácie sa ponúkajú dve metódy:

- *body prípadov použitia* (UCP)
- *plánovanie sústredené na architektúru* (ACSPP)

Dôležité je mať podrobnú špecifikáciu funkčných požiadaviek (pre UCP) alebo podrobný návrh architektúry (pre ACSPP). Architektúru systému môžeme jednoducho

odvodiť analógiou od existujúceho modelu architektúry. Zároveň je výhodné kombinovať postup zhora-nadol aj zdola-nahor.

Použitá literatúra

1. Bieliková, M.: *Manažment v softvérovom inžinierstve*. 1999.
2. Banerjee, G.: *Use Case Points: An Estimation Approach*, 2001.
3. Garmms, D., Herron, D.: *Estimating software earlier and more accurately.* The David Consulting Group.
4. Krašna, M., Rozman, I. Stiglic, B.: How to improve quality of software engineering management. *Software Engineering Notes*, Vol. 23, No. 3. ACM 1988. 120-125.
5. Paulish, D.J., Nord, R.K.: *Experience with Architecture-centered software project planning.* ACM 1996, 126-129

Annotation

Project estimation and work time planning in school software projects

Almost every software engineer started his active career with work on various school projects – whether they were projects for individuals or team projects. Although individual work is usually easier because man don't have to coordinate work of all team members, project planning and work time management is essential for both cases. It is important to accurately estimate the project in early stages of development, so that an accurate schedule can be created, because late delivery of the project often means total failure. In this essay I will deal with problems of software project planning and will provide some basic set of recommendations and advices with which the chance to successfully finish the software project will rise.