ČAS NA PLÁN B?

Keď Noe staval archu, ešte nepršalo.

Martin Mihalovič

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
xmihalovicm[zavináč]is[.]stuba[.]sk

Abstrakt. *Asi každý, kto bol niekedy súčasťou vývoja softvéru si uvedomuje dôležitosť plánovania takéhoto typu projektu. Asi každý, kto bol niekedy pri vývoji softvéru vie, že tu plány často nevychádzajú. Príčiny môžu byť rôzne, no všeobecne platí, že počítačový program je ťažko opísateľný (pokiaľ nie je hotový) a aj preto ťažko plánovateľný. V tejto eseji opisujem jeden spôsob, ako pri tvorbe plánu v ňom zahrnúť fakt, že v projekte je niečo, čo naplánovať neviem. Aj pri veľkom úsilí stále nie je zaručené, že projekt pôjde podľa plánu a nezmení sa na neriadenú strelu. Ak to nastane, je načase starý plán pochovať a nastaviť vývoj podľa nových pravidiel. Jeden pohľad na to, ako to urobiť, je tiež súčasťou tejto eseje.*

Kľúčové slová: *plánovanie, dva plány, zmena plánu*

Úvod

Keď sa spoločnosť vyvíjajúca softvérové systémy rozhodne vytvoriť nový produkt, ešte pred samotným vývojom zvyčajne urobí odhad času a nákladov, a podľa nich vytvorí plán, ktorým sa vývoj bude riadiť, určí tím, ktorý ho bude vykonávať a rozdelí v ňom úlohy. Počas tvorby programu sa priebežne kontroluje jeho stav voči plánu. Prípadné odchýlky sa operatívne riešia, aby mohol byť výsledný produkt odovzdaný včas, v požadovanej kvalite a v rámci rozpočtu. Neznie to rozprávkovo? Tí, ktorí sa už niekedy ocitli v tomto procese vedia, že takýto ideálny scenár býva zriedkavý. Je faktom, že softvérové projekty sú náchylné na prekročenie časových a finančných limitov. Tieto ťažkosti by možno v mnohých prípadoch nemuseli nastať, ak by bol plán nastavený lepšie.

V nasledujúcich riadkoch rozoberám jeden prístup k plánovaniu, ktorý vopred predpokladá ťažkosti. Aj keď je plánovaniu projektu venovaná pozornosť, jeho úplné

zlyhanie stále nie je vylúčené. O príčinách takéhoto zlyhania ako aj o tom, aké kroky by mohli vrátiť projekt späť k životu pojednávam v ďalšom texte.

Načo plány?

Aj keď si to asi neuvedomujeme, množstvo činností, ktoré denne vykonávame, je súčasťou plánu. Príkladom je pracovná doba, školský rozvrh alebo aj nastavenie budíka (naplánoval som si, kedy budem vstávať). Pri nepravidelných alebo zriedkavých činnostiach je plánovanie v podstate nutné. Ak sa napríklad rozhodnem ísť na výlet, musím vedieť, ako sa dostanem na dané miesto aj späť, čo tam budem robiť, čo budem jesť a podobne.

Prečo to ale robím? Nestačí, ak viem, čo chcem a spôsob dosiahnutia cieľa riešim „za behu"? Pri plánovaní rozmýšľam, ako v budúcnosti dosiahnuť nejaký cieľ čo najefektívnejšie. Zvažujem alternatívy a tú, ktorú považujem za najlepšiu, povýšim na plán. Pri tom sa mi väčšinou vybavia aj problémy, ktoré môžu nastať. Rozmyslím si, ako by som mal reagovať, keď sa objavia. Určím si teda, čo, kedy a ako budeme robiť a čo od toho očakávam. Výsledný plán sa v zásade nemusí líšiť od toho, ako by som konal bez neho. S plánom mám však istotu, že idem najkratšou cestou a znižujem šancu, že sa „zaseknem" v nečakanej situácii.

Plánovanie softvérových projektov

Softvérový systém sa podobá mnohým iným produktom v tom, že jeho vytvorenie sa riadi plánom. V takom pláne sú rozdelené úlohy medzi jednotlivých členov tímu, ktorí sa podieľajú na programovaní aplikácie, určuje kto má čo urobiť a v akom termíne. Výsledok je však opísaný len *vlastnosťami*, ktoré má spĺňať, konkrétna *forma* je vecou „umenia" programátora. To je rozdiel napríklad od stavebných plánov, ktoré priamo určujú podobu výslednej budovy (plán obsahuje presné rozmery, použitý materiál a pod.). Táto vlastnosť plánov softvéru súvisí s nemožnosťou jeho vizualizácie. Jediný spôsob, ako presne opísať program je až samotný zdrojový kód. Aj keď projektový plán môže obsahovať konkrétny opis hierarchie častí systému, jednotlivé moduly sú vždy určené len vlastnosťami. Toto je podľa mňa dobrý dôvod venovať plánovaniu softvérového projektu dostatočnú pozornosť a možno aj jedna z príčin, prečo sa plány v tejto oblasti častejšie odkláňajú od reality.

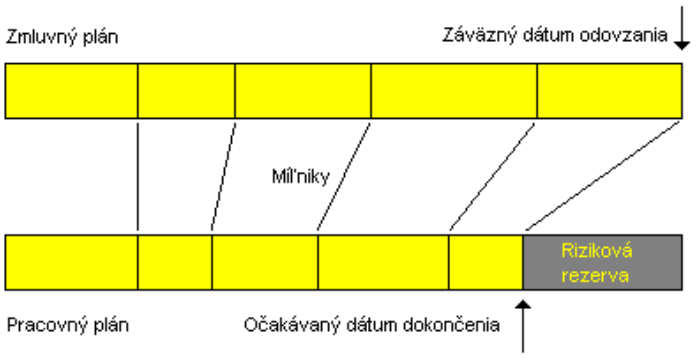
Ako teda plánovať softvér

Projektový plán softvérovej aplikácie okrem iného popisuje tri dôležité body vývoja:

- Rozvrh: určí sa celkový časový limit projektu a jednotlivé míľniky podľa ktorých možno sledovať súlad (alebo odklon) s časovým plánom.
- Rozpočet: množstvo financií určených na vývoj. Najväčšou položkou je ohodnotenie členov tímu. Pri prekročení finančného plánu (väčšinou ide ruka v ruke s nedodržaním rozvrhu) sa logicky navršuje aj cena výsledného produktu.
- Kvalita: úroveň splnenia funkčných požiadaviek na hotový program. Nedostatky kvality majú formu chýb (v jazyku programátorov tzv. bugov).

Vymyslené a do praxe zavedené boli viaceré metodiky a nástroje na odhadnutie potrebných vyššie uvedených zdrojov (z najznámejších COCOMO II, SLIM, SEER-SEM, PRICE-S a iné). V tejto eseji sa nimi hlbšie nezaoberám, boli dostatočne popísane v iných prácach.

Mňa zaujal jeden zvláštny prístup k plánovaniu, ktorý je viac rozšírením existujúcich techník (napr. menovaných vyššie) než samostatným plánovaním. Je popísaný v článku [1] a hovorí o vypracovaní hneď dvoch plánov: pracovného plánu (*work plan*) a zmluvného plánu (*commitment plan*). Prvý je tradičný projektový plán, podľa ktorého pracuje tím vývojárov. Vyjadruje, aké zdroje *odhadujem*, že bude projekt potrebovať v čase, keď sa plán tvorí. Zmluvný plán je ten, ktorý sa predkladá zákazníkovi. Zahŕňa pracovný plán plus vypočítané povolené prekročenie. Toto prečerpanie by malo vyjadrovať úroveň „neznalosti" budúceho systému v čase plánovania. Princíp dvoch plánov je znázornený na obrázku 1.

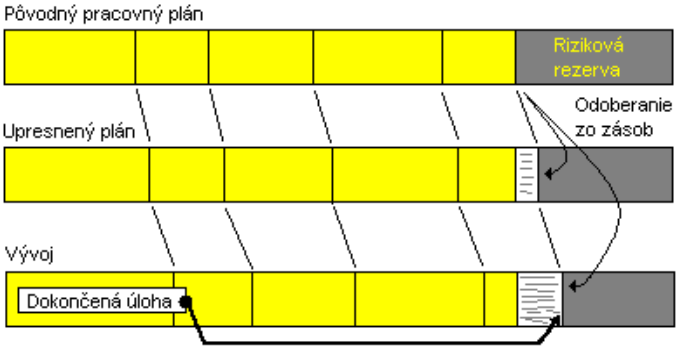


Obr. 1. Zmluvný a pracovný plán

Mnohých teraz napadne otázka: ako určiť rozdiel medzi plánmi? Na toto autor konceptu nedáva jasnú odpoveď. Riziková rezerva (*management risk reserve*) vyčísluje množstvo neurčitosti (*uncertainty*) v dátach, na ktorých je postavený pracovný plán. Myslím si, že jedným spôsobom môže byť identifikácia rizikových miest projektu a vytvorenia dvoch plánov: optimistického – predpokladám že všetko pôjde „ako po masle" a pesimistického – očakávam veľké množstvo problémov. Tieto dva plány sa potom stanú pracovným respektíve zmluvným plánom. Pomôcť by som si mohol aj predošlými projektmi; pozriem sa aké prekročenia plánov som mal pri iných programoch podobného rozsahu a/alebo zamerania a nimi sa inšpirujem pre rizikovú rezervu súčasného plánu.

Pri použití dvoch plánov by niekto mohol namietať ešte jeden háčik: mám dva dátumy dokončenia projektu – jeden pre tím a jeden pre zákazníka. Nie je tým rezerva vopred minutá? Alebo ju treba pred softvérovým tímom zatajiť? Tajenie časovej rezervy by som určite neodporúčal. Odhalenie pravdy by som ako manažér vysvetľoval tímu nerád. Dôležité je, že členovia tímu chápu význam rezervy ako znášanie rizika na strane softvérovej spoločnosti (bez nej by inak každé oneskorenie znášal zákazník) v podobe zdrojov vyhradených ešte tým, než sú potrebné. Napokon, ak som odhadol neurčitosť správne, vývoj sa bude predražovať a pomaly uberať z rezervných zdrojov.

Obrázok 2 ukazuje, ako to vyzerá v priebehu vývoja. Pôvodný odhad bol nízky. Ako sa dostávam viac do jadra problému, dokážem lepšie plánovať a znižujem ďalšie riziko. Je to za cenu odoberania zo zásob (*buying back the risk*). To je ale v poriadku, lebo s nižším rizikom potrebujem menej rezervy.



Obr. 2. Zmena pracovného plánu

Na tomto pohľade na odhadovanie a plánovanie ma oslovila najmä jedna vec: plánujem, že môj plán nebude celkom fungovať.

Minuli sa všetky rezervy. Čo teraz?

Aj plán (hoci aj dvojitý), ktorý sa javí v čase vzniku ako perfektný, môže totálne zlyhať. Príčiny môžu byť rôzne, no dokonalý plán podľa mňa ani nemôže existovať, pretože odhaduje niečo, čo nastane v budúcnosti. Krach softvérového projektu kvôli prečerpaniu naplánovaných zdrojov nie je pritom ojedinelý. Podľa [3] v rokoch 1999-2002 až 44% organizácií muselo odložiť alebo úplne zrušiť niektorý projekt pre podstatné prekročenie časových alebo finančných požiadaviek. Možno mnohé z nich nemuseli skončiť tak zle, ak by sa poriadne sledovalo plnenie plánu a pri prvých indíciách, že vystavený plán je nereálny a produkt nemôže byť podľa neho vytvorený, sa pristúpilo k jeho prestavbe. E. M. Bennatan vo svojej knihe [2] opisuje, kedy sa projekt rúti do katastrofy a navrhuje kroky k jeho návratu na požadované tempo.

Podstatným krokom je rozpoznanie, že projekt môže mať vážne problémy. Indikátorom sú nedostatky v troch pilieroch plánu spomínaných vyššie:

Rozvrh – proces má značné oneskorenie, ktoré sa navyšuje. Ako ale spoznám „značné“ oneskorenie? [2] navrhuje rozdeliť rozvrh na 12 častí a pozrieť sa na posledné 3. Ak počas nich meškanie neustále narastalo a zároveň celkový posun presahuje dĺžku troch období, je vhodné prehodnotiť plán.

Rozpočet – prekročenia rozpočtu väčšinou idú spoločne s prekročením času. Pre financie je stop kritérium jednoduché ako odpoveď na otázku „Je organizácia ochotná znášať takúto cenu?“

Kvalita – vývoj smeruje ku kritickému bodu ak zoznam chýb obsahuje množstvo tých kritických a za posledné tri obdobia sa nezmenšil alebo ak zákazník vyhodnocujúci program je silno nespokojný.

Toto je jeden spôsob, ako určiť zle naplánovaný projekt. Podľa mňa je dôležité komunikovať s budúcim používateľom, ak je to možné. Jeho negatívne reakcie na kvalitu vytváranej aplikácie sú dobrým varovným signálom možných problémov. Na druhej strane, ak už dochádza k prekročeniu vyhradených zdrojov, kontakt so zákazníkom mu môže pomôcť lepšie pochopiť príčiny problémov a akceptovať odklony od plánu. (Poznámka: agilné metodiky vývoja dokonca vyžadujú prítomnosť zástupcu zákazníka v tíme [4]). Napokon, cieľom projektu nie je držať sa plánu ale vytvoriť pre užívateľa hodnotný produkt.

Plán je mŕtvy, nech žije plán

Dobre teda, zistím, že pôvodný plán skôr potápa projekt a rozhodnem sa urobiť ústupky a nastaviť proces nanovo. Čo mám robiť? Má zmysel vôbec pokračovať? Odpovede sa snaží dať kniha [2] v podobe desiatich krokov, ktoré by mali prinavrátiť projekt na cestu k úspešnému dokončeniu:

1. *Zastaviť sa*. Je to rázny ale logický krok; ak starý plán stroskotal, nemá zmysel podľa neho pokračovať a prehlbovať problémy.
2. *Vybrať odhadcu*, čiže niekoho (projektového manažéra), kto určí skutočný stav projektu. Ideálny je nestranný človek, ktorý nebol súčasťou projektu.
3. *Vyhodnotiť stav projektu*. V tomto procese je vhodné zahrnúť aj vývojový tím. Zníži sa tým napätie a poznatky programátorov môžu byť cenné. Pri posudzovaní úloh, ktoré sa javia ako takmer hotové netreba zabúdať na pravidlo 90-50 (90% práce sa urobí za 50% času. Ďalších 50% času trvá dokončenie posledných 10% úlohy).
4. *Prehodnotiť tím*. Je to citlivá činnosť, ktorá by mala byť vykonaná taktne ale rázne. Je to súčasť vyhodnotenia a v tejto fáze žiadne zmeny v tíme nenastávajú. Sleduje sa technická zručnosť členov, schopnosti lídra viesť tím, tiež postoj členov voči projektu, či vzájomné konflikty.
5. *Určiť minimálne ciele*. V spolupráci s budúcim užívateľom (zákazníkom) sa nájdu najpodstatnejšie požiadavky. Všetky požiadavky sa rozdelia do troch skupín:
   - základné, bez ktorých sa produkt nezaobíde
   - dôležité, ktoré značne vylepšujú produkt ale nie sú základné
   - užitočné; zlepšenia, ktoré vo všeobecnosti nie sú dôležité

   Do nového plánu sa dostanú úlohy len pre prvú skupinu. Ostatné môžu prísť na rad až v nasledujúcich verziách.
6. *Dokážeme splniť minimálne ciele?* Toto je kritický krok v tom zmysle, že jeho výsledkom môže byť aj rozhodnutie v projekte nepokračovať. Je to otázka najmä smerom k manažmentu spoločnosti a zákazníkovi, či akceptujú pokračovanie projektu pri redukcii požiadaviek v predošlom kroku.
7. *Prebudovať tím*. Zmeniť sa môže štruktúra alebo aj zloženie tímu podľa hodnotenia z bodu 4.
8. *Analyzovať riziká*. Činnosť, ktorá sprevádza viaceré fázy vývoja softvéru, tu je však nevyhnutná. Identifikujú sa možné riziká, náhradné možnosti, priradí sa zodpovednosť za ich sledovanie.

9. *Urobiť nové odhady a rozvrh* na základe nových cieľov a tímu. Je rozumné zamerať sa na čas a kvalitu pretože aj dostatočný rozpočet nerieši problém ak je deň dodania nekompromisný.
10. *Vydať jasný prehľad projektu (míľniky).* Založenie časového plánu častých kontrol kvality zabezpečí, že projekt opäť nezíde z cesty.

Takýto nový plán má skôr núdzový charakter. Netreba však zabúdať, že ide o záchranu projektu, ktorému hrozilo zrušenie. Tento postup opäť vidím ako jednu možnosť, postupnosť sa môže líšiť v závislosti od zamerania a ohraničení projektu.

Pri preplánovaní projektu by som zdôraznil jednu vec, ktorá v predchádzajúcich bodoch priamo nevystupuje. Treba sa zamyslieť nad tým, *prečo* plán zlyhal. Ak viem určiť hlavnú príčinu (príčiny), v novom pláne si musím stanoviť, ako jej (im) predísť. Chyby, ktoré pochovali predošlý plán sa nesmú opakovať. Ak som napríklad predtým nesprávne odhadol finančnú náročnosť vývoja, potrebujem nájsť položky, ktoré spôsobili prekročenie rozpočtu a venovať im väčšiu pozornosť. Alebo ak za zastavenie procesu mohla nečakaná udalosť ako odchod členov tímu, zahrniem ju do rizík vo vynovenom pláne a stanovím si alternatívy.

Plánovanie špecifických projektov

Všetky doposiaľ opísané okolnosti plánov a plánovania sa vzťahujú najmä na „klasický" softvérový proces, čiže vývoj v profesionálnej organizácii, ktorá má svoju hierarchiu, platiaceho zákazníka a podobne. Existuje však aj množstvo osobitých projektov, pre ktoré je plánovanie nemenej dôležité. Napríklad písanie nekomerčného softvéru (prípadne open source), ak je robené vo viacčlennom tíme, malo by byť plánované a koordinované, plánovanie zdrojov však neobsahuje financie. Alebo ak je motívom vzniku aplikácie (a zároveň termínom spustenia) nejaká budúca udalosť, ktorá „nepočká" (napríklad zmena národnej meny), odklad dokončenia pravdepodobne neprichádza do úvahy.

Veľmi špecifickým typom projektu je zadanie z predmetu Tvorba informačného systému v tíme. Je osobitý z viacero dôvodov: po prvé, motiváciou študentov nie sú peniaze ale získanie skúseností so softvérovým procesom v tíme a aj získanie zápočtu. Po druhé: proces má jasne vyhradený časový rámec, odloženie sa nepripúšťa. Po tretie: rozloženie úloh v tíme a plánovanie celkovo je plne v réžii samotného tímu. Jednou podstatnou črtou projektu, ktorou sa podobá klasickým projektom je možnosť pravidelnej odozvy od „zákazníka", ktorého v tomto prípade predstavuje vedúci a zadávateľ projektu v jednej osobe. V takomto projekte by sa dal uplatniť koncept dvoch plánov; ak má tím celkovo približne 11 týždňov v jednom semestri, môže si vyhradiť rizikovú rezervu 1 až 2 týždne. Rozumné je rozdeliť si prácu na týždenné bloky a v každom kontrolnom bode sledovať naplnenie cieľov a konzultovať s vedúcim jeho spokojnosť s podobou produktu.

V prípade problémov s plánom je postup z predchádzajúcej časti použiteľný len v upravenej podobe. Takýto relatívne krátkodobý projekt nemôže dlho čakať a nemá k dispozícii žiadneho externého hodnotiteľa, zostavenie krízového plánu je v réžii tímu. Prehodnotenie a prestavba tímu sa týka len štruktúry, nemožno nikoho vylúčiť ani prijať. Najpodstatnejším krokom pri preplánovaní je dohoda s vedúcim na zmene požiadaviek tak, aby boli nové ciele dosiahnuteľné a zároveň akceptované obomi stranami. Z krokov

navrhovaných v [2] je teda vypustený krok 2, a 4 respektíve 7 sa týkajú len rozdelenia úloh v tíme.

Záver

Ktosi raz povedal: „Je ťažké robiť predpovede, najmä ak ide o budúcnosť." Je v záujme toho, kto plán vytvára, nielen naplánovať udalosti ale byť tiež pripravený, že udalosti zmenia plány. Na druhú stranu úspešný projekt nie je nutne ten, čo sa striktne drží plánu ale taký, čo vytvorí systém prínosný pre užívateľa, v rámci stanovených obmedzení.

Použitá literatúra

1. Armour G., P. : *To plan, two plans*. Communications of the ACM, Vol.48, Issue 9, 15-19, September 2005.
2. Bennatan, E. M.: *Catastrophe Disentanglement: Getting Software Projects Back on Track*, Addison-Wesley Professional, 2006.
3. Bennatan, E.M.: *The State of Software Estimation: Has the Dragon Been Slain? (Part 1)*, Executive Update Vol. 3, No. 10, The Cutter Consortium, July 2002.
4. Highsmith, J.: *Adaptive Software Development: A Collaborative Approach to Managing Complex System*, Dorset House, 2000.

Annotation

Time for plan B?

Everyone who has ever been involved in software development probably realizes an importance of planning in this kind of project. Everyone who has ever been involved in software development probably knows that the plans tend to be broken there. The reasons of it may vary, but generally it is not possible to capture a computer program (until not already done) and hence hard to plan. In this essay I describe a way of planning with consideration that there is something I'm actually unable to plan in the project. Even when tried hard, it is never guaranteed that a project will follow the plan and will not turn into an unguided missile. In such case it is reasonable to bury the old plan and set new rules to the development. An example of how it might be done is also part of this article.