Plánovanie v softvérovom projekte

PETER MIŠÁK

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
peter.misak@gmail.com

Abstrakt. Plánovanie softvérového projektu zahŕňa jednak tvorbu plánov, ich aktualizáciu, ale tiež monitorovanie ich plnenia. V softvérovom inžinierstve existuje mnoho názorov a prístupov k plánovaniu projektu. Pri použití bežných inžinierskych metód je potrebné tvorbe plánu venovať veľké úsilie a často sa stáva, že plán je ťažko aktualizovateľný a zmena plánu je buď nemožná, alebo môže mať na projekt vážne následky. Táto esej ponúka pohľad na tradičné metódy plánovania a konfrontuje ich s metódami, ktoré sa uplatňujú pri agilnom vývoji softvéru. Snaží sa vyzdvihnúť rozdielne nazeranie na plánovanie projektov v tradičných inžinierskych profesiách a v softvérovom inžinierstve, kde sa práve agilné metódy zdajú byť úspešnejšie.

V krajine, kde sa piesok lial...

Kde bolo tam bolo... Čo keby týmito slovami začínal zdokumentovaný plán softvérového projektu? Pripomína Vám to rozprávku? V rozprávke je možné všetko. Máme zázračnú guľu, ktorá nám vyveští budúcnosť a plánovanie je hračka. Určite by takýto plán bol dokonalý a, ako to už v rozprávkach býva, doviedol by softvérový projekt do úspešného konca.

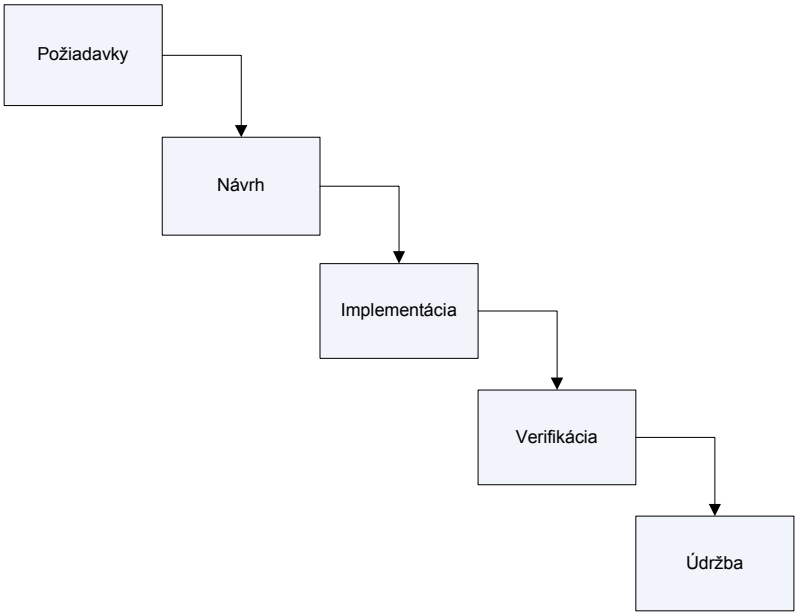
My však žijeme v reálnom svete. Žiadnu zázračnú guľu, ani palantír nevlastníme (aspoň ja osobne o nikom takom neviem), a predsa ten úspešný koniec túžime dosiahnuť tak my, ako i náš zákazník. Musíme si vystačiť s vlastným rozumom, odhadom a schopnosťami. V tejto eseji sa pokúsim zamyslieť sa nad problematikou plánovania v kontexte reálneho sveta a reálnych softvérových projektov.

... a kde sa sypal vodopád

Softvérové inžinierstvo je pomerne mladá disciplína, ktorej vznik sa oficiálne datuje od rokov 1968 – 1969. Postupne sa jeho súčasťou stávali rozličné procesy a metodológie vývoja softvéru. Už v roku 1970 vznikol tzv. *vodopádový model* vývoja softvéru, ktorý

Manažment v softvérovom inžinierstve, október 2006, s. 1-7.

mal zásadný význam vo vývoji softvéru v nasledujúcich rokoch a používa sa dodnes. Ide o sekvenčný model softvérového vývoja, ktorý ilustruje Obr. 1 [1].



Obr. 1. Schéma znázorňujúca vodopádový model.

Každý blok v obrázku znázorňuje jednu fázu vo vodopádovom modeli; najskôr sa získajú všetky detailné požiadavky od zákazníka, nasleduje návrh a analýza riešenia, implementácia, verifikácia a po nasadení systému jeho údržba.

Ako potom vyzerá plánovanie? Pravdupovediac, plánovanie v tomto prípade je priamočiare, pretože jednotlivé bloky na seba postupne nadväzujú. Tvorba plánu si síce vyžaduje veľké úsilie, avšak vytvorený plán môže pokrývať celý proces vývoja, prideľovanie úloh a v zásade nie je čo aktualizovať. Ak sa požiadavky v priebehu vývoja nemenia a nevznikajú chyby pri samotnom vývoji.

Skutočné projekty však často trpia nedostatkom špecifických požiadaviek od používateľa. Používateľ sám mnohokrát zistí, že to čo dostal, vôbec nie je to, čo chcel. V inom prípade môže dôjsť k tomu, že i keď výsledný produkt obsahuje vlastnosti, ktoré požadoval, tieto vlastnosti i tak nefungujú alebo fungujú zle. Plánovanie pri vodopádovom modeli nepočíta so zmenou a nepočíta s chybami – vecami, bez ktorých si náš život zrejme nevieme ani predstaviť. (To nemusí nutne znamenať, že zmena a náchylnosť človeka robiť chyby sú veci, ktoré po ktorých túžime, i keď o zmene to veľakrát aj skutočne platí. Chyby sú však ľuďom prirodzené a vznikajú tak u vývojárov, biznis analytikov, či návrhárov ako i u manažérov na strane klienta. Naším úsilím má byť chyby minimalizovať, avšak na druhej strane ich tiež brať do úvahy ako možné riziko.)

Ako sa vodopád zosypal

W.W. Royce vo svojej práci z roku 1970, načrtol sekvenčný model vývoja softvéru, ktorý nazval *vodopádový model*. Model sa dočkal veľkého ohlasu v softvérovom svete a našiel tiež uplatnenie v praxi. Paradoxné je, že samotný Royce o tomto modeli napísal, že je *„riskantný a náchylný na zlyhanie“* [1, 2]. Model, ku ktorému nakoniec dospel, mal bližšie k niečomu, čo dnes poznáme ako *špirálový model*.

Špirálový model však prvýkrát opísal Barry Boehm o pätnásť rokov neskôr. Boehm sa snažil do svojho konceptu zakomponovať dôležitosť všímania si rizík pri jednotlivých fázach vývoja. Vo svojom článku z roku 1985 tvrdí, že úplne použitie tohto systému vývoja zvýšila produktivitu u všetkých sledovaných projektov o 50% [3]. Náčrt modelu je vidno na Obr. 2.

Obr. 2. . Ilustrácia následností fáz v špirálovom modeli životného cyklu tvorby softvéru.

Jednotlivé fázy sa opakujú v iteráciách a dôraz sa kladie na rozpoznávanie rizík a vyhodnocovanie rôznych alternatív na základe miery rizika. Plán teda nie je možné do detailov spracovať na začiatku projektu a vyžaduje si pravidelné aktualizácie a prehodnocovanie. Ako postupovať pri plánovaní takto vyvíjaných projektov? Inšpiráciou nám môže byť spôsob PADRE (plánovanie - **p**lan, odsúhlasenie - **a**pprove,

vykonanie - **d**o, revízia a korekcia - **r**eview and revise a vyhodnotenie - **e**valuation). Princíp PADRE spočíva v nasledujúcom postupe [4]:

- Rozdelenie celého projektu na viaceré etapy, pričom každá etapa predstavuje funkčnú časť výsledného projektu (dĺžka etapy môže byť napríklad 4-5 mesiacov)
- Rozdelenie každej etapy na fázy:
  - Vytvorenie makety – veľmi hrubého modelu, ktorý môže byť prezentovaný potenciálnemu koncovému používateľovi
  - Tvorba funkčného prototypu – maketa slúži ako špecifikácia
  - Dolaďovanie a zlepšovanie na základe návrhov zákazníka, resp. koncových používateľov
  - Údržba

V každej etape sa vývoj rozdelí na menšie moduly, pričom zodpovednosť za každý modul je pridelená konkrétnemu človeku. Takýto prístup poskytuje ľahšie zmeny v plánovaní konkrétneho modulu.

Ak sú moduly jasne definované, najkompetentnejší členovia tímu pridelia jednotlivým modulom stupeň náročnosti (ťažký / stredne ťažký / ľahký). Rozdelenie modulov slúži pri odhadovaní času, ktorý je potrebný pre vývoj daného modulu. Do úvahy je potrebné vziať napríklad rozsah zložitosti implementácie daného riešenia, pričom použitých môže byť viacero metrík (počet potrebných tried a metód, niekedy sa tiež odhaduje počet riadkov kódu a pod.). Odhadnutý čas a závislosti medzi modulmi je vhodne zaznačiť formou Ganttovych a PERT diagramov. Nakoniec musí byť jasné, že celý tím súhlasí s plánom predtým, ako sa proces vývoja na moduloch začne. Pri samotnom vývoji modulov sa potom v cykle aplikuje metóda PADRE:

- *Plánovanie* je splnené vtedy, ak jeden z členov tímu vypracuje plán implementácie modulu a poskytne ho na okomentovanie ostatným členom tímu. Platí zásada, že čím väčší tím, tým menšia možnosť participácie všetkých jeho členov.
- *Odsúhlasenie* spočíva v oboznámení celého tímu s plánom, jeho pripomienkovaní a dohode na jeho konečnej verzii.
- *Vykonanie* zrealizuje programátor implementovaním príslušného modulu
- *Revízia a korekcia* je potrebná, ak modul nefunguje podľa predstáv a je treba vykonať dodatočné testovanie a upravovanie
- *Vyhodnotenie* nastáva keď je už modul hotový a vedúci tímu s programátorom hodnotia vykonanú prácu a analyzujú prípadné zlepšenia

Plánovanie a aktualizácia plánov, ako aj prideľovanie úloh pri špirálovom modeli sa môže diať v jednotlivých cykloch, teda iteráciách a zdá sa byť ľahšie realizovateľná ako v prípade použitia klasického vodopádového modelu. Jednotlivé plány sa dajú

lepšie zrealizovať, keďže sa pracuje s rozsahovo menším problémom (nie projekt, ale modul).

Plánovanie v modernom softvérovom projekte

V súčasnosti sa čoraz viac začínajú vo vývoji softvérových produktov uplatňovať tzv. agilné metódy. Existuje viacero metodológií, ktorým sa môže dať prívlastok „agilné“. Veľmi populárnym je napríklad extrémne programovanie (XP), Crystal, či SCRUM.

Vo všeobecnosti prevláda názor, že tieto metódy sú použiteľné najmä v prípade menších tímov a projektov, pri ktorých nie sú od začiatku známe konkrétne požiadavky. Avšak, osobne si myslím, že pre väčšinu projektov je vhodné použiť model menších tímov, je však potrebné väčšie projekty modularizovať a rozdeliť na samostatné časti, pokiaľ je to len možné. I keď sa zdá, že požiadavky sú definované pomerne presne, veľmi často dochádza k ich zlej interpretácii alebo pochopeniu.

Agilné metódy vychádzajú z myšlienky iteratívneho spôsobu vývoja. Zmena je vítaná a veľký dôraz sa dáva na testovacie procedúry, tzv. unit testy. Dôležitosť správneho a fungujúceho kódu má prioritu pred dôležitosťou dokonalej dokumentácie. Ako je to s plánovaním?

Mohlo by sa zdať, že tomuto druhu vývoja softvéru by sa najviac hodilo neplánovať vôbec. Nie je to však pravda, pretože minimálne zákazník chce mať predstavu o vývoji produktu, o tom, koľko za to má zaplatiť a kedy to dostane. Navyše, pre tím je veľmi dôležité mať hĺbkovú predstavu o vývoji z krátkodobého hľadiska. Jedným z možných riešení je vydať sa cestou rizikom riadeného plánovania [5], kedy sa pri každej iterácii vytvoria možné plány a na základe rizík sa vyberie jeden z nich. V raných fázach by to mali byť plány s relatívne väčším rizikom, aby sa zvýšilo porozumenie potrieb zákazníka, v konečných fázach naopak, s čo najmenším rizikom, aby sa stihlo produkt odovzdať načas. Môžeme nadviazať na plánovanie pri špirálovom modeli vývoja softvéru, pričom definujeme ešte zopár zásad [6]:

- *Plánovať do správnej hĺbky* – robia sa detailné krátkodobé plány, vágnejšie dlhodobé.
- *Vyjednávať rozsah projektu* v prípade, že sa projekt nestíha – tu sa uplatňuje princíp agilného vývoja, že fixná je cena a čas, hýbať je možné len s rozsahom.
- *Zákazník určuje prioritu* – teda čo a kedy má byť implementované.
- *Plán odzrkadľuje realitu* – tvoria sa detailné plány iterácie, z ktorých musí byť jasné, čo sa bude robiť a kto je za čo zodpovedný; je dôležité klásť dôraz aj na spätnú väzbu zákazníka z predchádzajúcej iterácie, nové požiadavky a opravy chýb.
- *Spätná väzba je životne dôležitá* – spätná väzba od zákazníka môže eliminovať množstvo rizík; je potrebné myslieť na plánovanie prototypov, či ukážok funkcionality už v raných fázach vývoja.

- *Tri typy vydaní produktu* – interný, pre potreby testovania, skúšobný, ktorý môže zákazník okomentovať a napokon produkčný, ktorý predstavuje finálny produkt.
- *Plánovanie prerábania kódu* – agilné metódy vývoja môžu byť náchylné na tvorbu chaotického a neprehľadného kódu, je preto potrebné naplánovať zmeny v kóde, vedúce k jeho sprehľadneniu a vylepšeniu.
- *Zváženie kladov a záporov agilného vývoja* – každý projekt je individuálny a je preto potrebné zvážiť, nakoľko agilný a nakoľko „tradičný" má byť prístup k vývoju (niekedy je možné predísť neskoršiemu prerábaniu kódu skorším dokonalejším návrhom).
- *Zváženie vážnych rozhodnutí v raných fázach vývoja* – príkladom zlého rozhodnutia, ktoré zásadným spôsobom môže ovplyvniť produkt aj pri agilnom vývoji, je implementovanie podpory jedného používateľa, pričom zákazník v podstate požaduje viacpoužívateľský prístup k systému. V takomto prípade je zmena v neskorších fázach tak zásadná, že môže znamenať prerobenie všetkých častí produktu. Je potrebné dôkladne zvážiť každé rozhodnutie a zákazníka informovať o možných dôsledkoch zmeny v kľúčových prípadoch.

Vidno, že agilné metodológie, používajúc princíp iterácií, kladú dôraz hlavne na konkrétne a detailné „mikroplánovanie", úzky kontakt so zákazníkom a taktiež úlohu jednotlivca v tíme. Uprednostňované je prevzatie úlohy členom tímu pred jej pridelením projektovým manažérom (člen tímu má zodpovednosť za úlohu prevziať, zodpovednosť mu nemá byť nanútená). Participácia členov tímu na plánovaní na začiatku iterácie a vyhodnocovaní na jej konci sa môže prejaviť kladne v synergickom efekte a v pocite, že každý člen tímu môže ovplyvniť projekt a prispieť k jeho úspechu. Predpokladom je, že členovia vývojového tímu sú kreatívni ľudia a preto nazeranie na nich, ich schopnosti a zodpovednosti, musí byť iné ako nazeranie na robotníkov v továrni [7]. A nakoniec, úzka spolupráca so zákazníkom, prototypovanie, konzultácie navrhnutých používateľských rozhraní a funkcionality, to všetko pomáha eliminovať riziko neskorších zmien v požiadavkách. Samozrejme, aj tu platí, že zákazník musí chcieť.

Pár slov na záver

Je vodopádový model mŕtvy? Je potrebné rozvíjať a propagovať plošné používanie agilných metód plánovania? Vodopádový model mŕtvy nie je. NASA, či Ministerstvo obrany Spojených štátov amerických s úspechom používajú vodopádový model na vývoj softvéru [8]. Zástancov má aj medzi populárnymi softvérovými inžiniermi. Príkladom je známy softvérový „komentátor" Joel Spolsky [9].

Vývoj v posledných rokoch, ako i moje osobné skúsenosti s tvorbou softvéru v tíme, ktorý adoptoval agilnú metodológiu pri práci na dvojročnom projekte pre

Iránske ministerstvo telekomunikácií ma však presviedčajú, že plánovanie má zmysel, ak ráta so zmenou. Od situácie závisí, akú dôležitosť pripíšeme detailným dokumentom a kolosálnym plánom a akú pripíšeme plánovaniu testovania, hodnotenia, účasti zákazníka na vývoji. Je treba však na pamäti, že nežijeme v rozprávke, ale v dynamicky meniacom sa svete plnom chybných rozhodnutí.

Použitá literatúra

1. Royce, W.W.: Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, Vol. 26, no. August (1970), 1-9.
2. Kolektív autorov, *Waterfall Model*, http://en.wikipedia.org/wiki/Waterfall_model, 25.10.2006
3. Boehm, B.W.: A Spiral Model of Software Development and Enhancement, *Computer*, Vol. 21, No. 5 (May 1988), 61-72.
4. Rettig, M., Simons, G.: A project planning and development process for small teams, *Communications of the ACM*, Vol. 36, Issue 10 (October 1993), 45-55.
5. Mingshu, Li, Meng Huang, Fendgi Shu, Juan Li: A risk-driven method for eXtreme programming release planning, *Proc. of the 28th international conference on Software engineering,* Shanghai (2006), 423-430.
6. Collins-Cope, M.: *Plannnig to be Agile?*, http://www.softwarereality.com/lifecycle/PlanningToBeAgile.jsp, 12.11.2003
7. Fowler, M.: The New Methodology, http://www.martinfowler.com/articles/newMethodology.html, 13.10.2005.
8. Kolektív autorov: The Waterfall Model, *Parametric Cost Estimating Handbook*, http://www1.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm, 14.9.1995.
9. Spolsky, J.: The Project Aardvark Spec, http://www.joelonsoftware.com/articles/AardvarkSpec.html, 17.8.2005

Annotation

Software project planning

Software project planning consists of plans creation, actualization and monitoring of how they are fulfilled. There are many opinions and ways of project planning, in software engineering. Traditional engineering methods require great effort of plan creation and it is not uncommon that plan becomes impossible to actualize. Change of plan in such situations is either impossible or can affect project itself critically. This paper presents overview of planning for standard engineering methods on one hand, while giving the reader description of agile software development planning on the other hand. It tries to highlight the differences between these two approaches, while accentuating agile methods as they have become popular and seem to be successful in many cases.