Metódy plánovania softvéru

JÁN PERHÁČ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
perhac05[zavináč]fiit[.]stuba[.]sk

Abstrakt. Úspech plánovania v softvérových projektov záleží na schopnostiach človeka porozumieť vzťahu medzi úlohou a jej časovou neurčitosťou a túto neurčitosť správne vyzualizovať.V softvérovom inžinierstve existuje veľa metód, ktoré sa snažia túto neurčitosť do väčšej či menšej miery eliminovať. Projekty použitím takýchto metód sa považujú za robustnejšie a menej náchylné voči neočakávaným udalostiam. Vo svojej eseji sa zameriavam na porovnávanie a hodnotenie metód plánovania. V skratke opisujem, na čo sa využívajú a pri akých typoch projektu je vhodné ich použiť. Snažím sa o ohodnotenie ich prínosov v plánovaní a zároveň sa snažím odhaliť ich nedostatky, ktoré v sebe ukrývajú. Ako prostriedky na porovnávanie mi slúžia: schopnosť používateľa naučiť sa danú metódu, spôsob vizualizácie a aké informácie v sebe ukrývajú. Dôležitým aspektom pri hodnotení všetkých metód je aj schopnosť používateľa vyčítať z nej potrebné informácie.

Plán náš každodenný

Všetci vieme, respektíve si myslíme, že vieme, čo je to plán a ako sa takýto plán vytvára. Už od útleho detstva sa stretávame s plánmi, ktoré chceme alebo musíme dodržiavať. V škôlke máme presne určený čas na spanie, školský rozvrh nás privádza do zúfalstva a keď sa ho už konečne zbavíme, tak príde na rad tvrdá realita a plány si už musíme vytvárať sami. Štátny rozpočet, rozpis futbalových zápasov, kalendár a mnoho iného sú tiež plány. Plánuje sa a bude sa plánovať všade.

Čo je vlastne plán? Je to séria úkonov, ktoré je potrebné vykonať na dosiahnutie určitého cieľa. Takúto definíciu dostaneme, ak toto slovo budeme hľadať v slovníku. Dosiahnutie cieľa je veľkou motiváciou, keď vykonávame akúkoľvek činnosť. Takouto činnosťou je aj softvérové inžinierstvo, preto je pochopiteľné, že plán a samotné plánovanie v ňom zohárava veľkú úlohu.

V prípade softvérového inžinierstva je cieľ určitý softvér a úkony sú všetky akcie, ktoré je potrebné vykonať pred vytvorením softvéru. Všetci tý, ktorí vytvárajú plány softvérových projektov to majú o to ťažšie, že postup pri vývoji softvéru nie je

Manažment projektov softvérových a informačných systémov, október 2008, s. 1-8.

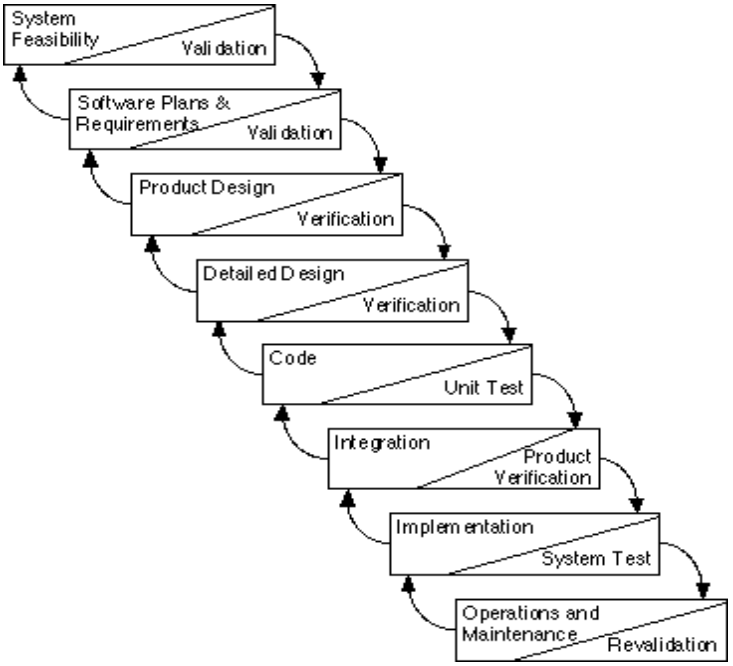
jednoznačný. V tomto procese je príliš veľa premenných, ktoré ovplyvňujú dĺžku a rozsah takéhoto projektu.

Začalo to vodopádom

Ako sa vyvíjal softvér, tak sa vyvíjali aj postupy ako tento softvér navrhnúť a implementovať. Spočiatku sa tieto postupy veľmi líšili. Počítače sa vyskytovali iba na určitých miestach a ľudia, ktorí na nich pracovali mali len veľmi malé skúsenosti. Dôležitú úlohu v tomto období zohral aj fakt, že počítače sa dodávali hlavne do škôl a vedeckých inštitúcií. Keďže v týchto organizáciách pracovali ľudia systematicky, tak sa tieto postupy snažili implementovať aj pri softvérovom inžinierstve.

Prvým všeobecne uznávaným modelom vývoja softvéru bol vodopádový model[6], tento model vychádza z predošlých modelov, ktoré používali určité fázy vývoja. Tento model sa vo veľkej miere využíval v 70. rokoch, kedy ho jeho autor predstavil ako určitý koncept, ako by sa malo postupovať pri vývoji. Jeho hlavným princípom je pokračovať v ďalšej fáze až po ukončení predchádzajúcej. Postup vodopádového modelu je na Obr. 1, ktorý bol prevzatý z [2].

Aj keď mal tento model veľa odporcov a v dnešnej dobe sa už prakticky nevyužíva, je podľa môjho názoru akýmsi základným kameňom všetkých modelov a metód. Zadefinoval všetky fázy, ktoré sú potrebné pri vývoji a zároveň poukázal aj na to, že je nevyhnutné používať metodológiu v softvérovom inžinierstve.



Obr. 1. Vodopádový model

Ako prax ukázala, tak tento model bol a je pre väčšinu projektov nevyužiteľný. Postupom času sa začali používať iné modely. Väčšina z týchto modelov sa snažila vyhnúť nedostatkom vodopádového modelu a zároveň využiť jeho výhody.

Takýmto významným modelom je aj evolučný model. Tento model sa využíva, keď požiadavky na daný softvér nie su vopred dané. Počas vývoja sa tieto požiadavky špecifikujú a v ďalších verziách sa implementujú. Takýto postup je prospešný hlavne pre zákazníka, ktorý môže celý proces ovplyvňovať podľa svojich požiadaviek. Tak ako každý model, aj tento má svoje nevýhody. Jednou z nich je aj nedostatočná modularita. Každá ďalšia verzia je založená na predošlej. Takýmto spôsobom môže dôjsť k situácii, keď požadovaná zmena v produkte už nie je možná.

Ďalším zaujímavým modelom je špirálový model, ktorý je podrobne opísaný v [2]. Jeho výhoda spočíva v jeho flexibilite. Ak je malé riziko, že sa navrhne zlý program a jeho používateľské rozhranie, ale je veľké riziko, že sa prekročí rozpočet, tak sa tento model začne správať podobne ako vodopádový. Ak je to naopak, tak sa správa ako evolučný model. Problémom v tomto softvéry je réžia, ktorá je potrebná na jeho manažovanie.

Z ďalších modelov by som ešte spomenul Vmodel, ktorý je postavený na vodopádovom modely. Iteratívny model vytvorí v každej iterácii produkt, ktorý prešiel všetkými fázami vývoja. Znovu použitím funkčných častí softvéru sa zaoberá komponentový model. Špeciálnou skupinou sú takzvané agilné metódy. Tie by sa dali opísať ako plánovanie za pochodu. Nevytvára sa komplexný plán, ale iba určité úlohy, ktoré sa majú vykonať v blízkej budúcnosti.

Všade samé problémy

Boli by sme asi naivní, keby sme si mysleli, že sa plánovanie zaobíde bez komplikácií. Plánovanie v softvérovom inžinierstve má určité svoje špecifiká, preto sa len veľmi ťažko vytvára plán, podľa ktorého by sa aj reálne postupovalo. Existuje viacero štúdií, ktoré poukazujú na problémy plánovania. Jedna taká štúdia je opísaná v [8].

Z tejto štúdie vyplývajú tieto problémy plánovania:

- Plánovanie zdrojov: Zdroje v pláne sú často neúplné, viacznačné, nedostatočne zadefinované.
- Plánovanie nákladov: Odhadnúť náklady spojené s vývojom je takmer nemožné. Väčšina projektov na nedostatočný rozpočet.
- Plánovanie hodnotenie projektu: Metódy na hodnotenie sú nepostačujúce, čo vedie k zlej kvalite softvéru alebo sú nadhodnotené, čo vedie k ilúzii, že softvér nedosiahol požiadavky.
- Plánovanie projektu: Plánovanie v softvérovom inžinierstve je vo všeobecnosti slabé.
- Plánovanie rozvrhov: Pretože určiť dĺžku vykonávania jednotlivých úloh je ťažké, sú aj rozvrhy často krát chybné.

- Plánovanie testovania: Neexistuje univerzálny spôsob testovania v softvérovom inžinierstve. Platí, že bezchybné testovanie nezaručí bezchybný softvér.
- Plánovanie kontroly: Metódy, ktoré pomáhajú manažérom kontrolovať projekt sú nedostatočné.

Bojujeme s problémami

Ako som už spomenul, nástroje na riešenie máme, otázne je ako efektívne ich dokážeme využívať a do akej miery nám pomôžu. Všetky metódy, ktoré sa využívajú, majú stanovené postupy a spôsoby ako ich uplatňovať. Na určité problémy sa uplatňuje iba určitá metóda alebo skupina metód. Tú nie je možné použiť na iné problémy.

Tak ako prebieha projekt, tak prebieha aj plánovanie. Má určité postupy ako vytvárať plán. Počas tohto procesu sa zadefinujú požiadavky na plán. Zadefinujú sa odpovede na 3 základne otázky: kto, čo a pre koho. Toto väčšinou vychádza z podstaty projektu. Máme zákazníka, pre ktorého je tento projekt a máme tým vývojárov, ktorí sa o to pokúsia. Čo sa ma vyrábať zadefinuje objednávateľ. Ešte poznamenám, že objednávateľ nemusí byť nevyhnutne zákazník.

Definovanie cieľa

Po definovaní týchto bodov si musíme určiť ciele, ktoré chceme dosiahnuť. Tie budú slúžiť ako motivácia počas celého projektu, preto treba mať na pamäti, že musia byť určené presne. Jednou z metód je aj SWOT analýza odvodená od anglických slov sila(Strengths) , slabosť(Weaknesses), príležitosti(Opportunities), hrozby(Threats). Používa sa na odhalenie silných a slabých stránok tímu a taktiež na zistenie príležitostí a hrozieb, ktoré by mohli tím postihnúť[3]. Z vlastnej skúsenosti viem, že bez poznania samého seba nie je možné dosiahnuť ciel. Ak je tím vedomý si svojich silných a slabých stránok vie si zadefinovať ciele, ktoré sú reálne a dosiahnuteľné. Ak však tím nie je dostatočne kritické k sebe, tak ciele bývajú nadhodnotené a ich nesplnenie vedie k zníženiu morálky tímu a poklesu produktivity.

Ďalšou metódou, ktorá sa využíva pri určovaní cieľa je SMART (Specific, Measurable, Agreed, Relevant, Timebound) metóda. Ako už názov hovorí ide o inteligentné určenie cieľa. Vytvorenie takéhoto cieľa nezaberie veľa času, ale môže naopak nejaký čas alebo pridanú hodnotu získať. Príkladom je využitie konkrétnej technológie v projekte. Cieľ je zadefinovaný tak, aby sa daná technológia využila, pretože predstavuje pridanú hodnotu. Ak by cieľ nebol zadefinovaný podrobne, je veľká pravdepodobnosť, že sa využije technológia, ktorá je známa a nie tá, ktorá predstavuje pridanú hodnotu.

Plánovanie úloh

Ďalším krokom pri plánovaní je dekompozícia úloh na čo najmenšie úlohy, ktoré sa budú dať vykonať v určitom čase. Overenou metódou je WBS (Work Breakdown

Structure), ktorá sa využíva v mnohých projektoch aj mimo softvérového inžinierstva. Základným princípom je vytvoriť si graf, kde koncové uzly predstavujú úlohy. Krásou tohto grafu je, že vieme presne určiť kto bude za ktorú úlohu zodpovedný. Problémom takejto metódy je zdvojenie úloh. Pri dekompozícii môžeme objaviť úlohy, ktoré sú totožné, hoci ich názvy sa líšia. Opačný problém je ten, keď sú úlohy zdanlivo totožné, ale ich realizácia si vyžaduje rozdielne prístupy.

Vyzualizačným prostriedkom na plánovanie úloh je tiež matica úloh. Každý člen tímu v tejto tabuľke ľahko nájde svoje meno a úlohy mu pridelené.

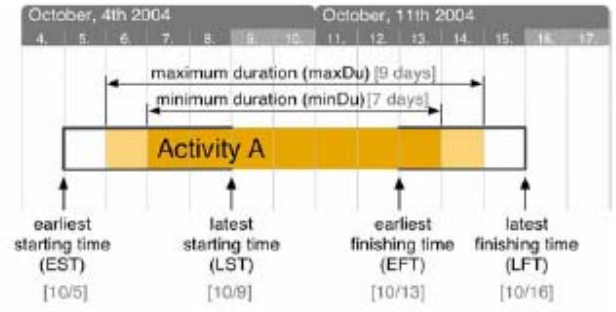
Plánovanie rozvrhov

S predchádzajúcou témou úzko súvisia aj rozvrhy. Poznáme už úlohy, ktoré sme pridelili členom tímu, ale neviem jednu podstatnú vec a to je čas, ktorý majú na tieto úlohy vyhradený. Základnom je určiť si čas, do kedy sa majú úlohy vykonať. Na jednoduchú, ale efektívnu vyzualizáciu slúži Gantt diagram. Jeho výhody a hlavne nevýhody popísali Lung-Chun Liu a Ellis Horowitz vo svojom článku[4]. Gantt sa vyznačuje prehľadnosťou úloh. Jeho najväčšou slabinou je, že úlohy majú presne daný čas a nepočíta s neočakávanými udalosťami. Používateľ nadobúda dojem, že projekt sa skončí vtedy, keď sa ukončí posledná úloha.

Metóda CPM(Critical Path Method) je založená na orientovaných grafoch. Úlohy si zoradíme tak, aby predstavovali graf, kde hrana predstavuje náveznosť úlohy. Pomocou tejto metódy sa dá určiť čas, za ktorý sa vykonajú všetky úlohy. Kritická cesta predstavuje postupnosť úloh, ktoré je potrebné vykonať na ukončenie projektu. Takouto metódou odhalíme úlohy, ktoré sa môžu pralelne vykonávať, bez ohľadu na to, ktorú úlohu na kritickej ceste vykonávame. Problém nastane vtedy, ak vytvorený graf má príliš veľa vetvení a tiež, keď sú cesty rovnocenné.

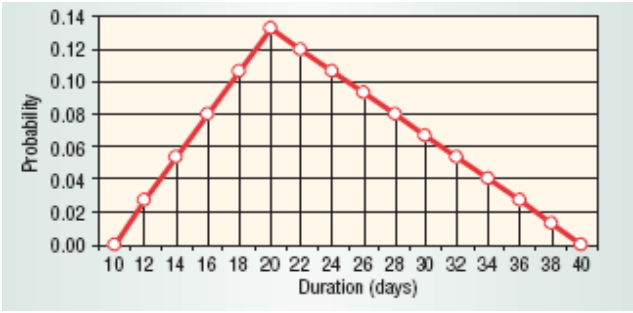
Úspech plánovania v softvérových projektov záleží na schopnostiach človeka porozumieť vzťahu medzi úlohou a jej časovou neurčitosťou a túto neurčitosť správne vyzualizovať. Premenlivosťou úloh sa zoberajú autory v knihe[7]. Metóda PERT(Program Evaluation and Review Technique) je zameraná na neurčitosť a je ďalším stupňom metódy CMP. Obsahuje obdĺžniky a šipky, kde obdĺžniky predstavujú plohy a šipky predstavujú vzťahy medzi nimi. Presné časové informácie sa nachádzajú vnútry úloh a sú to konkrétne dĺžka trvania, najskorší začiatok a najneskorší začiatok. Takto spracované úlohy nám dávajú reálnejší obraz o plnení úloh. V porovnaní s Gantt diagramom ale predstavujú väčšiu náročnosť na spracovanie údajov o čase. Preto sa využívajú na zobrazenie kritickej cesty a zvyšné úlohy sú graficky zakreslené do Gantt diagramu.

Autori nedávno vyvinutej metódy PlanningLines vo svojom článku[1] opisujú túto metódu a hodnotia ju v porovnaní s PERT metódou. Názorná ukážka s popisom PlanningLines je na obrázku 2. Podľa experimentov vykonaných na vzorke študentov majú obe metódy skoro rovnú výpovednú hodnotu. PlanningLines prevažovali pri určovaní dĺžky vykonávania, začiatku a konca aktivy, PERT zase pri otázkach ohľadom konktrétnej úlohy.



Obr. 2. Planning Lines

Správny manažér si je vedomí toho, že čas potrebný na splnenie úlohy záleží od viacerých okolností. Ak sa všetky priaznivé predpoklady splnia na 100 percent a nepriaznivé na 0 percent, tak úloha sa vykoná za najkratší čas. Ak sa naplnia na 100 percent nepriaznivé predpoklady, tak úloha sa vykoná za najdlhší čas, v krajných prípadoch sa nevykoná. Na obrázku 3 je vidieť pravdepodobnosti s akými sa vykoná úloha v danom čase.



Obr. 3. Pravdepodobnostný graf

Štatistickým prístupom riešenia otázky plánovania sa zaoberá SPID(Statistically Planned Incremental Deliveries)[5]. Táto metóda je komplexnejšia a rieši už aj samotné vykonávanie plánu a dodávky softvéru. Použitím štatistických metód sa spomedzi spomenutých metód približuje najviac realite, preto je z môjho pohľadu asi najvhodnejšou. Postup pri tejto metóde je zozbieranie potrebných informácií a naplánovanie prvého cyklu tak, aby jeho vykonanie bolo s pravdepodobnosťou 95 percent úspešné. Ďalšie cykly sa plánujú podľa prvého cyklu. Táto metóda sa zaoberá aj odmeňovaním ako prostriedkom motivácie. Za každý ďalší cyklus je vypísaná odmena. Plán na druhý a tretí cyklus sa vytvára s pravdepodobnosťou 50 percent. Vývojári si takto uvedomujú, že pri väčšej snahe budú aj adekvátne odmenení.

Pár slov na záver

Pokrok v plánovaní softvérových projektoch je proces, ktorý, pevne dúfam, bude pokračovať. Veľmi ma teší, že každým rokom pribúdajú nové a nové metódy, ktoré túto činnosť do väčšej či menšej miery zjednodušujú. Dôležitým cieľom pre všetkých softvérových inžinierov by mala byť snaha osvojiť si tieto metódy a snažiť sa ich implementovať vo svojich projektoch. Iba takýmto spôsobom sa dozvieme, ktoré metódy sú výhodné a ktoré treba prepracovať a aj takýmto spôsobom prispejeme k lepšiemu softvéru.

Použitá literatúra

1. Biffl, S., Thurnher, B., Goluch, G., Winkler, D., Aigner, W., Miksch, S.: An Empirical investigation on the Visualization of Temporal Uncertainties in Software Engineering Project Planning. Computer, v.38 n.5, 437-446, Máj 2005
2. Boehm, B.: A Spiral Model of Software Develpment and Enhancemet. Computer, v.21 n.5, 61-72, Máj 1988
3. Houben, G., Lenie, K., Vanhoof, K.: A knowledge-based SWOT-analysis system as an instrument for strategic planning in small and medium sized enterprises, Decision Support Systems, v.26 n.2, 125-135, August 1999
4. Liu, L., Horowitz, E.: A Formal Model for Software Project Management. Transactions on software engineering, v.15 n.10, 1280-1293 Október 1989
5. Miranda E.: Planning and Executing Time-Bound Projects. Computer. v.35 n.3, 73-79, Marec 2002
6. Royce, W. W.: Managing the Development of Large Software Systems. Proc. 9th. Intern. Conf. Software Engineering, IEEE Computer Society, 1987, 328-338 Originál vydaný v Proc.WESCON, 1970.
7. Tatnall, A., Shackleton, P.: IT Project Management: developing on-going skills in the management of software development projects. International Conference on Software Engineering: Education and Practice. 400-405, Jún 1996
8. Thayer, R.H., Pyster, A., Wood, R.C.: Validating Solutions to Major Problems in Software Engineering Project Management. Computer, vol. 15, no. 8, pp. 65-77, Aug., 1982

Annotation

Software planning methods

The success of software projects depends on the ability of a human planner to understand the relationships of tasks and their temporal uncertainty an hence the visualization thereof. In software engineering are many methods which are eliminating the task uncertainty. Projects which use these methods are considered as robust and more efficient. In this essay I am describing planning methods, comparing to each other and trying to find its advantages and

disadvantages. I am comparing them from several perspectives. For example, one perspective is effort which is needed to read important information from these methods.