Pozícia analýzy vo vývoji softvéru

LUKÁŠ BELEŠ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
beles05[zavináč]fiit[.]stuba[.]sk

Abstrakt. Vo svojej eseji by som ozrejmil dôležitosť vedy Softvérové inžinierstvo vo svete. Začal som popisovať situáciu z histórie, lebo vtedy sa programovalo inak ako dnes. Neskôr opíšem situáciu súčasnú, ktorú som rozdelil na dve epochy. V prvej epoche som vysvetlil vodopádový model životného cyklu, s ktorým mám aj ja skúsenosti, lebo táto metóda sa používa najmä v škole. A ako druhú epochu agilného programovania som vysvetlil extrémne programovanie, ktoré sa podľa mňa stane najpoužívanejšou metódou v budúcnosti. Keďže softvérové inžinierstvo je len teória tak v nasledujúcej časti to vysvetlím z praxe. Budem sa zameriavať na jednotlivých členov tímu. Každý ma iné úlohy a každý prináša iné rizika do projektu.

Úvod

V súčasnosti, v dobe veľkého rozmachu informačných technológii, sú veľké požiadavky na vývoj softvéru. Ako pre každú vedu, tak aj pre softwarové inžinierstvo existuje veľa pracovných postupov, metód a odporúčaní. Na nasledujúcich riadkoch by som chcel popísať aké vývojové metódy existujú. Pri vytváraní nového projektu je veľmi dôležité vybrať si správnu vývojovú metódu tak, aby bola výhodná rovnako pre nás, ako aj pre zákazníka. Nedá sa povedať, že existuje optimálna vývojová metóda. Pre každý projekt je to špecifické. Preto si myslím, že je veľmi dôležité oboznámiť sa so všetkými možnosťami softvérového inžinierstva. Hlavne je to dôležité vedieť pre projektových manažérov, alebo pre človeka, ktorý je vedúcim vývoja. Prečo je to dôležité? Odpoveď na túto otázku je jasná: peniaze a čas. Ako následok nevhodnej vývojovej metódy sa môže náš projekt nielen že časovo predĺžiť, ale zároveň zvýšia sa náklady na daný proces.

Každá metóda prináša niečo pozitívne a zároveň niečo negatívne. Myslím si, že novšie metódy sa „ponaučili“ zo starších metód, čoho dôsledkom je, že prinášajú viac pozitívneho ako negatívneho. Chcel by som túto problematiku porovnať aj s minulosťou, lebo bola iná situácia vývoja ako je dnes. Hlavným cieľom bude porovnať ich na základe skúseností, ktoré som nadobudol počas štúdia a počas zamestnania. Tejto oblasti sa venujem už veľmi dlho, a mal som možnosť použiť

Manažment projektov softvérových a informačných systémov, október 2008, s. 1-7.

viaceré vývojové metódy pre vývoj softvéru. Je dôležité si uvedomiť, že chyby vo vývoji nerobia metódy a postupy, ale robia ich ľudia. Preto v poslednej časti sa pozriem na človeka - programátora z psychologickej stránky. Pozriem sa na všetky pracovné pozície v tíme, lebo každá si inak definuje plán, a každá iná pracovná pozícia vytvára iné riziká.

História

História softvérového inžinierstva nie je stará. Je to jedna z najmladších zo všetkých vied. Jej začiatky si ja však nepamätám. Bohužiaľ. Ale dobre si ich pamätá môj oco. Keďže v tej dobe programoval malé programy pre jednu nemenovanú fabriku. Teraz už nemá na to čas, a už dlho nevytváral projekty. Práve v dôsledku dlhého časového sklzu máme dosť rozdielne názory na vývoj softvéru. Minule sa ma pýtal, že na čo sú nám analytici. Vtedy taká pozícia neexistovala a tvorilo sa inak ako sa tvorí dnes.

Aké metódy sa používali a aké rizika so sebou prinášali? Treba si uvedomiť najskôr, že na aké účely sa vtedy programovalo a tvorilo. Boli to veľmi malé a jednoduché programy. Logika programu bola jednoduchá. Vedelo sa aký bude vstup a aký bude výstup. A práve vstup a výstup bol vždy jednoznačný. Napr. keď som v škole programoval vymazanie adresára na úrovni strojového kódu, tak som vedel že vstup môže byť len meno adresára. A výstup bude len odpoveď či som úspešne vykonal svoju prácu. A práve takéto jednoduché programy sa vytvárali. Moje riešenie som nemusel integrovať do žiadneho existujúceho systému. Nemusel som konzultovať so žiadnym ďalším človekom, lebo som vedel že môj program bude izolovaný od všetkého. Jediné na čo som bol závislý bol operačný systém a hardware. Práve kvôli jednoduchej logike nebolo potrebne veľa konzultácie so zákazníkom resp. zadávateľom práce. Neexistovalo detailne plánovanie, všetko bolo priamočiare a jasné. Horšie to bolo práve z technického hľadiska, keďže programátor musel vedieť veľa technických vlastností procesora alebo iných zariadení v osobnom počítači. Musel poznať vlastnosti operačného systému.

Myslím si, že v minulosti sa programovalo lepšie ako dnes. Programátor bol naozaj programátorom. Dnes sa čím ďalej, tým viac kladie dôraz na administratívne veci, čo dosť zaťažuje programátora. Tým som nechcel povedať, že vtedy neboli žiadne riziká pri vývoji. Ba naopak, boli. Ale boli odlišné ako sú dnes. Najväčšie riziko predstavovala práve technická znalosť hardwaru. Dosť často sa mi stalo, že program ktorý fungoval na jednom počítači nefungoval na inom, lebo tam bol iný procesor.

V 60. rokoch 20. storočia vypukla softvérová kríza. Projekty sa robili veľmi dlho, boli veľmi nekvalitne naprogramované. Nestíhali sa termíny. Ba dokonca veľa projektov sa nedokončilo. Keď si porovnám metódy, aké sa vtedy používali a aké sa používajú dnes, tak je jasné, kde bola chyba. Neexistovala žiadne teória o tom, ako vytvárať väčší projekt. Žiadna koordinácia a komunikácia v tíme.

Podľa môjho názoru, hlavná chyba bola v analýze. Analyzovalo sa veľmi málo. Bez analýzy sa dnes ani nepohnem. Musím prebrať všetky možnosti a riziká s tým

spojené. A hlavne vedieť si všetko naplánovať. Aby som si vedel určiť termíny, kedy budem čo robiť.

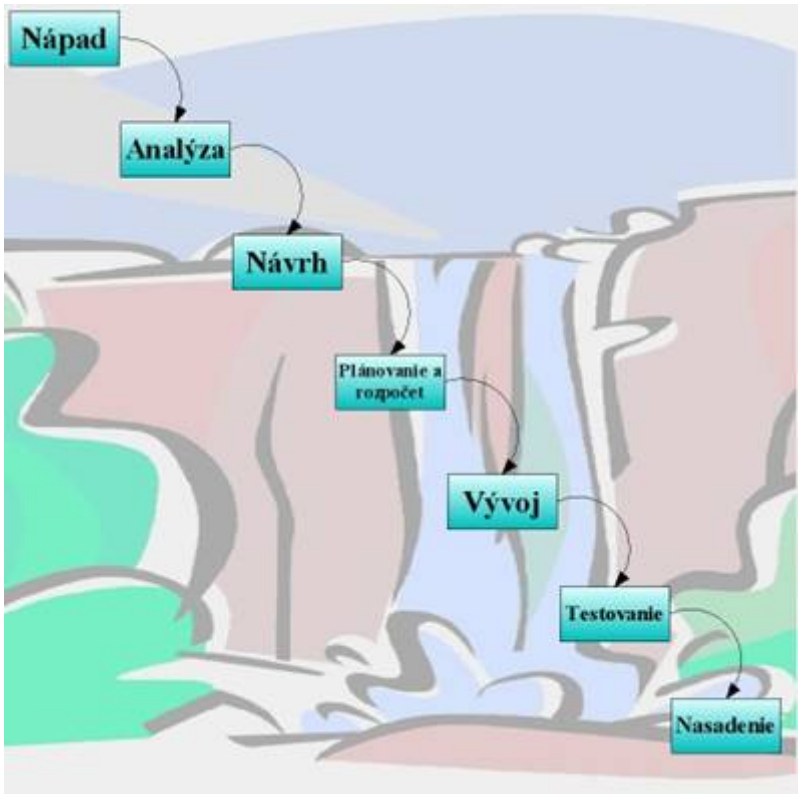
Tak zle som programoval keď som mal asi 15 rokov. Zobral som zadanie. Žiadna analýza, plán. Sadol som si za počítač a začal som vytvárať aplikáciu. Po nejakom čase som narazil na nejaký problém, ktorý som nepredvídal a nakoniec som strávil niekoľko hodín opravou programu. Nepoužíval som žiadne testy a ak som spravil nejakú zmenu, tak zmena sa prejavila negatívne v inej časti programu. Dalo by sa povedať, že som bol tiež v softvérovej kríze ako boli profesionáli v 60 rokoch 20. storočia. Čím viac som program rozširoval o nové funkcie, tým viac sa komplikovala údržba, lebo som program zle navrhol. Takéto približne symptómy mala aj daná softvérová kríza. Časom som sa priučil a začal som používať metódy, ktoré sa používajú dnes.

Súčasnosť

V súčasnosti je situácia omnoho komplikovanejšia ako bola v minulosti. Skoro žiadny projekt nepracuje ako hodinky. Termíny sa nestíhajú. Výrobná produkcia projektu sa predžuje. Ako hlavný problém vidím v zlom naplánovaní jednotlivých častí projektu. Tým myslím od analýzy a až po údržbu.

Existuje nevraživosť voči kompetentným orgánom Európskej únie, že nechcú prideliť peniaze z európskych fondov na projekty. Podľa môjho názoru, konajú správne. Lebo vedia ako to dnes beží. Dialóg skončí vždy len kvôli tomu, že chýba detailne plánovanie projektu alebo detailný návrh projektu. Ako ďalší fakt by som uviedol nevedomosť zákazníka. Dalo by sa povedať, že veľa krát ani nevie, čo presne chce. A až počas vývoja projektu mu to začne myslieť podrobnejšie, čo ma za následok časté zmeny v návrhu alebo v analýze a projekt sa zbytočne predlžuje. Práve preto vzniklo veľa jazykov na komunikáciu so zákazníkom. Ako je napr. BPEL[2]. Je to jazyk, pomocou ktorého vedia spolu komunikovať analytici a zákazníci, pričom ten jazyk je spoločný, ale jeho používanie je ešte v plienkach. A bohužiaľ na Slovensku ho používa veľmi málo firiem.

V súčasnosti by sme softwarové inžinierstvo rozdelili na epochy vývoja. A to na tradičné metódy a na agilné metódy. Medzi tradičné metódy patria napr. vodopádovy model životného cyklu projektu a jeho derivácie. Dobrým príkladom agilného smeru je napr. Test Driven Development, alebo Extrémne programovanie. Mám skúsenosti s obidvoma typmi vývoja softwaru. Preto viem porovnať obidve oblasti. Prečo? Tradičné metódy používame najmä v škole a agilné metódy sa presadzujú v praxi. Tu by som videl podstatný rozdiel vývoja softvéru. Myslím si, že akademická verejnosť trocha zaostáva v oblasti softvérového inžinierstva. Veľa študentov, ako napríklad aj ja po skončení školy netuší, čo je agilné programovanie. V škole sa učíme resp. robíme zadania (pozri Obr. 1). Čo je typicky príklad *vodopádového modelu životného cyklu projektu*[1].



Obr. 1. Vodopádový model životného cyklu projektu

Nevytvárame projekt po krokoch. Veď na také malé projekty aké robíme v škole je tento model vyhovujúci, lebo je úplne jednoduchý. A však pre väčšie projekty by priniesol veľa rizík, lebo nedokáže obhospodáriť rozsiahlejšie a komplikovanejšie aspekty. Ako ďalšie riziko by som uviedol jeho nepružnosť. Do vývoja nie je možné vstupovať tak aktívne. Všetko je striktné podľa postupnosti fáz vývoja. Pokiaľ príde zákazník s hocijakou zmenou v zadaní, je nutné znovu prejsť na fázu analýzu, návrh, znova zmeniť všetky dokumenty a podobne. Z vlastných skúseností viem, že pokiaľ by som robil aplikáciu pre komerčného zákazníka podľa tohto modelu, tak by som softvér vyvíjal aspoň o rok dlhšie. Lebo dodanie produktu zákazníkovi by prebiehalo formou veľkého tresku. So zákazníkom treba komunikovať nie len na začiatku projektu, teda pri objednávaní zákazky. Ale aj počas vývoja. Počas vývoja mu ukazovať jednotlivé časti projektu, aby sa vedel ku všetkému vyjadriť a my to potom s ním skonzultovať. Z vlastných skúsenosti viem, že keď mi niekto povie na konci, že chce pridať ešte jednu tabuľku, tak dodanie projektu sa predlží o niekoľko dní. Zákazník si myslí, že je to jednoduchá vizuálna záležitosť, ale pritom on nevidí, že treba zmeniť analýzu, návrh, implementáciu a testovanie.

Chybu by som hľadal aj u zhotoviteľa projektu. V jeho záujme by mala byť efektívna komunikácia so zákazníkom. Aj keď veľa projektových manažérov to využíva ako spôsob ako „ošklbať“ zákazníka o peniaze vďaka jeho neznalostiam. To je jedno z najväčších rizík pre zákazníka.

Podstatné plánovanie vývoja softvéru vždy nastavuje analytik alebo projektový manažér. Analytik sa niekedy opýta aj programátora, že ako dlho to bude trvať, ale analytik si to vždy upraví podľa toho ako sa dohodne zo zákazníkom. Čo zapríčiňuje to, že programátor musí niečo spraviť rýchlejšie ale zároveň nekvalitnejšie ako za normálnych okolností.

Mňa v praxi najviac zaujalo *Extrémne programovanie*[3] čo je jeden typ z agilného vývoja softwaru. Programátori robia odhad ceny a času na projekte, a vďaka ním sa spresní termín odovzdania a cena projektu. To umožňuje stanoviť priority a rozhodnúť, čo je skutočne potrebné vyriešiť a čo je druhoradé. Čo na druhej strane môže spôsobiť to, že programátor nediskutuje priamo so zákazníkom. Nevie čo je práve priorita pre zákazníka. Lebo dosť často to nevie ani on sám posúdiť. Rýchly výstup - programátorsky tím zabezpečí prvý funkčný vzor tvoreného v systéme v čo najkratšom čase a v krátkych cykloch robí jeho rozširovanie so zohľadnenými pripomienkami používateľa. Tento fakt platí len pre skúsených programátorov. Pokiaľ je takáto úloha zadaná začiatočníkovi, môže trvať implementácia dlhšie ako pri dôkladnej analýze. Okrem toho rýchly výstup neošetruje vždy všetky chyby, ktoré je potom nutné opraviť. Extrémne programovanie je známe tým, že testy sa robia skôr než implementácia. Ja si myslím, že vďaka tomu je implementácia čistejšia a viac ošetrená voči chybám, aj keď v dobe písania testu neviem niekedy overiť všetky možné výstupy.

Pochopiteľne, aj spôsob písania kódu je dôležitý. A to pravé Extrémne programovanie striktne prikazuje. Je dôležité si na začiatku povedať názvovú konvenciu premenných. Správne formátovanie kódu. Správne názvy metód alebo procedúr. Aby sa človek lepšie orientoval v kóde. Potom kód vyzerá tak, že ho vytvoril jeden človek.

Koniec teórie, prax je iná

Ako som v úvode naznačil softvérové inžinierstvo je len teória. Prax je (ako vždy) úplné iná. Lebo ak vieme, že pre daný projekt je najlepšia metóda napr. RUP (Rational Unified Process) ale ho neovládam, tak ho silou mocou ani nepresadzujem, lebo to nezvládnem.

Je rozdiel ak daný projekt rieši skupinka ľudí teda tím, alebo daný projekt rieši jeden človek. Pokiaľ človek robí sám, berie celé riziko na seba. Plán si navrhuje sám, nemusí s nikým konzultovať okrem zákazníka. Sám si rozloží sily a čas. Najväčšie riziko predstavuje to, že je na to sám a nevie všetky potrebné fakty. Môže sa dostať do takého bodu vývoja, s ktorým nevie pohnúť ďalej. Napr. človek ktorý vie dobre programovať, nevie robiť dobre s databázami.

Práca v tíme je lepšia voľba pri väčších projektoch. Každý sa špecializuje na svoju oblasť, a nemusí vidieť do inej oblasti. Ako najväčšie riziko vidím v komunikácii

medzi členmi v tíme a zosúladenie úloh v tíme. Každý člen tímu ma inú výkonnosť. Vždy bude niekto čakať na niekoho. Tým sa dĺžka projektu predlžuje.

Pohľad ľudí v tíme

Aj keď je dnes veľký dopyt po informatikoch, často nájdeme v tíme ľudí, ktorý nemajú takú prax a skúsenosť ako by bola od nich žiadaná. Vo viacerých tímoch nájdeme veľa brigádnikov alebo študentov. Študent nemá žiadne skúsenosti. Predstavuje nejaké riziko pre projekt alebo tím? Myslím si, že nie. Z mojich skúseností viem, že väčšinou sú tam len „do počtu". Ich prácu by mohol nahradiť hocikto iný v tíme. Je to výhodné pre projekt ale aj pre samotného brigádnika. Nejaká tá skúsenosť sa na neho nalepí a zarobí si pekné peniaze. A projekt si vykáže viac človekohodín. Spôsob plánovania ľudí je zaužívaný všade, len nie v sfére informatiky. Horšie by bolo keby na danom projekte robil zodpovednú prácu. Ak je na brigádnickej dohode, nikto ho nemôže donútiť pracovať viac a hlavne nie je spoľahlivý.

Najväčšie riziko v tíme predstavujú analytici alebo projektoví manažéri. Nemali by to byť ľudia introverti. Mali by veľmi dobre komunikovať s ľuďmi v tíme a so zákazníkmi. Okrem toho veľmi dobre plánovať a navrhovať prácu. Ale ak však niečo meška, vždy je chyba u nich.

Programátor môže byť človek introvert, ale aj extrovert. Nemusí komunikovať so zákazníkom. Ale mal by komunikovať s ľuďmi v tíme. Mal by vedieť správne si naplánovať dĺžku práce na úlohe, aby to vedel niekto z kompetentných ľudí zosúladiť. Dôležité riziko je nedokonalosť písania kódu: neprehľadnosť, chybovosť, neotvorenosť, nepružnosť, zlé názvoslovie premenných a podobne. Ak bude niekto upravovať kód po niekom, kto má tieto zlé vlastnosti, tak čas vyriešenia chyby bude podstatne dlhší.

Záver

Vývoj softvéru nie je jednoduchý. V práci preto zdôrazňujem najmä dôležitosť naštudovania softvérového inžinierstva, aby programátor vedel, aké metódy sa používajú a vedel ich potom správne priradiť k projektom. Každá metóda má iné plánovanie a iné riziká. Pokiaľ analytik vytvára tím, mal by tiež poznať psychológiu človeka a poznať ľudí, ktorí na tímovom projekt majú participovať. Pre efektívne riešenie úlohy by mali mať zvládnuté odborné znalosti, ale tiež vedieť komunikovať v tíme, mať chuť pracovať a učiť sa nové veci.

Použitá literatúra

1. Clear, T.: The waterfall is dead.: long live the waterfall!!. In: *ACM SIGCSE Bulletin,* Vol. 35, No. 4 (2003) 13-14.

2. Lohman, N., Massuthe, P., Stahl, C., Weiberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation. In: *Data & Knowledge Engineering,* Vol. 64, No. 1 (2008) 38-54.
3. Sfetsos, P., Angelis, L., Stamelos, I.: Investigating the extreme programming system---An empirical study. In: *Empirical Software Engineering,* Vol. 11, No. 2 (2006) 269-301.

Annotation

The position analyze in developed software

In my essay I would like tell us importance science Software engineering in the world. At the beginning I describe situation from history, because in this time programming was other than today. Later I describe actual situation, which I divide on two epochs. In the first epoch I explain waterfall model living cycle. This one I have experience, because this method is using at school. As the second epoch agility programming I explain extreme programming, which I will suppose become the best method in the future. Seeing that software engineering is only theory, so next part I explain it from praxes. I will focus on separately element of team. Everybody has own task in team and everybody bring other risk on the project.