Dokonalý plán

TOMÁŠ KUZÁR

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
tomas.kuzar[zavináč]gmail[.]com

Abstrakt. Plánovanie má v rámci softvérového projektu nezastupiteľnú úlohu. Vývoj každého softvérového produktu je svojim spôsobom jedinečný, preto nie je dostatočné definovať plán ako postupnosť krokov, ktoré majú viesť k stanovenému cieľu. V počiatočných fázach nie je možné jednotlivé kroky detailne opísať, navyše svet softvérového inžinierstva je veľmi dynamický a plán musí byť dostatočne robustný nato, aby sa vedel so zmenami vyrovnať. Preto sa esej zameriava na tvorbu plánu na úrovni architektúry organizácie, resp. na úrovni usporiadania tímu. Esej sa nesnaží poskytnúť návod, ako vytvoriť dokonalý a presný plán, ale na vhodné zadefinovanie štruktúry organizácie a v nej fungujúcich procesov tak, aby zaručovali splnenie vytýčených projektových cieľov načas a v požadovanej kvalite.

Úvod

S problematikou plánovania sa stretávame v mnohých oboroch ľudskej činnosti. Pri jednoduchších činnostiach možno plán chápať, ako popis jednotlivých krokov, ktoré vedú k želanému cieľu. Plánovanie v softvérovom inžinierstve však predstavuje netriviálny problém, keďže softvérové projekty sú vo väčšine prípadov pomerne rozsiahle a na ich začiatku je veľká časť úskalí neviditeľná. Z tohto dôvodu je ambícia vytvoriť dokonalý plán veľmi odvážna.

Definovaním vhodného plánu budeme vedieť odpovedať na otázku, čo má byť urobené a kedy je to potrebné urobiť. Na druhej strane, definovaním procesov určíme, ako to má byť urobené a v akej kvalite. Inak povedané, podľa [4] podstata a hlavná úloha procesu projektového plánovania spočíva v stanovení cieľov projektu a spôsobov vedúcich k ich dosiahnutiu.

Plánovanie a procesne orientovaný prístup

V literatúre [6] sa je možné dočítať, že najlepšou východiskovou pozíciou pre začatie projektu je jasne definovaný plán a príslušné procesy. V úvode mojej eseje by som

chcel predstaviť pohľad na plánovanie, ktorý sa opiera o špirálový model vývoja softvéru a sústreďuje sa na definovanie procesov.

V rámci tohto pohľadu na plánovanie je možné projekt rozdeliť do trochu úrovní: projektová úroveň, úroveň etáp a úroveň modulov. Zodpovednosť na projektovej úrovni nesie základný tím, na úrovni etáp projektový tím a na úrovni modulov samotní programátori. Na každej úrovni použijeme špirálový model, ktorý pozostáva z piatich základných aktivít: naplánuj, odobri, vykonaj, skontroluj, vyhodnoť. Týmto spôsobom je teda zadefinovaná základná kostra postupu prác.

V prvom rade je potrebné zadefinovať jednotlivé procesy na úrovni projektu. Projekt je vhodné rozdeliť na etapy, ktoré možno chápať ako menšie projekty. „Menší projekt sa riadi jednoduchšie ako veľký“ [6]. Jedná sa teda o vyššiu úroveň riadenia projektu a spomenúť možno niekoľko základných procesov, ktoré by mali byť na tejto úrovni uskutočnené:

- sformovanie základného tímu a napísanie základných cieľov projektu
- popísanie projektu: meno projektu, účel, špecifikácia požiadaviek
- vytvorenie projektového tímu a určenie rolí v tíme
- rozdelenie projektu na etapy
- prehodnotenie jednotlivých krokov

Každá etapa pozostáva z modulov. V [6] definujú modul ako časť práce, ktorá musí byť na projekte vykonaná. Je teda potrebné jednotlivé moduly identifikovať, popísať a následne určiť objem práce, ktorý si konkrétny modul vyžaduje. Základné kroky na úrovni etáp:

- rozdelenie projektu na moduly
- určenie závislosti a časové následnosti medzi modulmi
- pridelenie modulov jednotlivým členom
- vyhodnotenie uskutočnených krokov

Zodpovednosť za modul preberá programátor, je to jeho vlastný malý projekt a on sám je zodpovedný za jeho odovzdanie v stanovený čas v dostatočnej kvalite. Základné kroky na úrovni modulov sa opierajú priamo o špirálový model vývoja a zahŕňajú už spomenutých päť krokov: naplánuj, odobri, vykonaj, skontroluj a vyhodnoť. Práca na samotnom module je teda v kompetencii programátora.

V tejto kapitole je opísané plánovanie projektu založené na špirálovom modeli a na definovaní jednotlivých procesov. S týmto prístupom môžem princípe súhlasiť, avšak tento pohľad nerieši problém priradzovania zodpovednosti za jednotlivé moduly a tiež sa nezaoberá mnohými rizikami, ktoré by mohli vytvorený plán významne ohroziť. Preto neskôr v eseji opíšem ďalšie prístupy, ktoré môžu byť vhodným doplnením opísaného procesne orientovaného plánovania.

Vyššie opísaný postup je pomerne statický a často nedostatočne reflektuje požiadavky reálneho dynamického sveta. Plánovanie v softvérovom inžinierstve je obzvlášť náročná úloha, lebo dopredu nie je možné presne identifikovať všetky úskalia – rozsah a zložitosť niektorých modulov, zmenu špecifikácie, zmenu technológie

a ďalšie komplikácie rôzneho druhu. To znamená, že len definovanie jednotlivých krokov a procesov, ktoré majú viesť k riešeniu je dobrá, nie však bezchybná cesta vedúca k dokonalému plánu.

Skupiny softvérových projektov a plánovanie

Prístup k plánovaniu rozoberaný v predchádzajúcej kapitole sa zameriaval na tradičný vývojársky projekt, teda na projekt, ktorý má základné etapy vývoja – špecifikácia, návrh, implementácia a testovanie. Existuje aj množstvo iných projektov, ktorá do tejto základnej schémy nespadajú – údržba, servis, prispôsobenie produktu, vývoj riadený plánom a mnohé iné. To teda znamená, že mnohé projekty majú spoločné črty a nie je potrebné vždy vytvárať plán odznova. Tento prístup nesie v sebe však aj mnohé riziká. V [**Error! Reference source not found.**] sa môžeme dočítať: „Mnohé prvky môžu byť použité znovu, ale plánovači musia starostlivo zvážiť, ktoré prvky plánu predchádzajúceho projektu je možné aplikovať aj v novom projekte."

Môžeme sa dopracovať k dvom extrémom – vytvárať vždy nový plán presne podľa požiadaviek projektu alebo vytvoriť jeden všeobecný plán a aplikovať ho na každý projekt. V prvom prípade nebudeme vždy presne a jasne vedieť zadefinovať potrebné procesy, lebo pôjde vždy o niečo úplne nové, v druhom prípade budeme mať jasne zadefinované procesy, ktoré však nebudú reflektovať potreby konkrétneho projektu. Preto za vhodné riešenie považujem rozdelenie projektov do skupín na základe ich podobného zamerania.

Projekt si vyžaduje vykonávanie činností, ktoré sú pre danú skupinu projektov príznačné. Podľa môjho názoru tento prístup môže byť vhodným doplnením procesne orientovaného modelu. Rozdelením projektov do skupín podľa ich podstaty, budeme jasnejšie vedieť popísať jednotlivé procesy a tiež ľahšie identifikujeme mnohé riziká. Ale keďže v rámci tímového projektu ide v drvivej väčšine prípadov o tradičné vývojárske projekty, v ďalšom texte sa budem venovať výhradne im.

Pohľad na plánovanie v zmysle softvérovej architektúry

Na začiatku realizácie veľkých projektov je prakticky nemožné navrhnúť plán iba definovaním postupnosti krokov a aktivít. Nie je totiž možné dostatočne presne predvídať, ktoré aktivity bude potrebné vykonať. Preto je vhodné v prvom rade navrhnúť architektúru softvéru a až na základe kompletnej architektúry vytvoriť harmonogram prác na projekte. Architektúra môže byť vyhotovená hneď po kompletizácii špecifikácie požiadaviek. „Plán a plánovanie zdrojov bez návrhu vhodnej architektúry nemá prakticky žiadnu hodnotu" [**Error! Reference source not found.**]. Podľa mňa je táto skutočnosť pri veľkých projektoch zrejmá. Nie je možné pustiť sa do rozsiahleho softvérového projektu bez určitej predstavy o jeho architektúre. Jednotlivé kroky tohto prístupu chápem ako všeobecný návod, ako pri takomto projekte postupovať.

Vývoj teda začína návrhom riešenia s vysokou mierou abstrakcie. Základné kroky pri tvorbe plánu vychádzajúce z architektúry podľa [**Error! Reference source not found.**]:

- vysoko úrovňový abstraktný dizajn
- harmonogram práce zhora nadol
- odhady komplexnosti komponentov – návrh zdola nahor
- plány verzií
- harmonogram projektu
- harmonogram vývoja
- osobné harmonogramy

Plánovanie vychádzajúce z architektúry má však aj svojich odporcov, ktorí v [**Error! Reference source not found.**] hovoria, že hoci architektúra môže pomôcť znížiť cenu očakávaných prispôsobení, je tam významná neistota, či tieto prispôsobenia budú reálnou odpoveďou na dopyt trhu. Preto autori navrhujú sústrediť sa pri vývoji vždy na konkrétny modul a nebrať pri tom ohľad na jeho budúce verzie a iné zmeny, ktoré môžu ale nemusia v budúcnosti nastať. Osobne tento prístup nepovažujem všeobecne za prospešný. Pri vývoji riešenia by mali mať aspoň niektorí členovia tímu určitú všeobecnú víziu o budúcnosti daného produktu.

Plánovanie podmienené architektúrou môže byť vhodne skombinované s procesným prístupom k plánovaniu, ktoré som opísal v predchádzajúcej časti eseje. Architektúra bude považovaná za základnú kostru projektu. Vhodná predstava o architektúre bude predpokladom pre rozdelenie celého systému do menších častí a navyše starostlivo zadefinovaná architektúra umožní jednoduchšie systém v budúcnosti rozširovať.

Metodika CCM a zlepšovanie procesov plánovania

Metodika kladie veľký dôraz na určenie rozsahu projektu a na určenie potrebného úsilia. Niektorí považujú určenie rozsahu projektu za najdôležitejší problém vo vývoji softvéru. Mnohí vedci a inžinieri podľa [**Error! Reference source not found.**] určenie rozsahu a potrebného úsilia považujú technický problém. Opierajú sa o viacero metrík, ktorá môžu byť použité na vyjadrenie rozsahu projektu – počet obrazoviek, počet tabuliek, počet riadkov kódu a podobne. „V malých projektoch je možné vyjadriť rozsah projektu aj na základe predchádzajúcej skúsenosti programátorov“ [**Error! Reference source not found.**]. Odvolávajúc sa na [**Error! Reference source not found.**] technický pohľad je vhodný pri kontrole vývoja.

Nezanedbateľnou skutočnosťou pri zlepšovaní procesov je podľa [**Error! Reference source not found.**] učenie sa z predchádzajúcich chýb. Hoci je tímový projekt jediným predmetom svojho druhu počas štúdia, mnohé postupy som si osvojil aj na iných predmetoch len v menšom rozsahu. Presné definovanie procesov má významný vplyv na kvalitu výstupu.

Po detailnom zadefinovaní procesov je možné vniesť do projektov rutinu. Schémy opísané v predošlých častiach eseje je možné chápať ako kostru projektu. Kvalitne zadefinované a optimalizované procesy nám umožnia odpovedať na otázku, ako vytvoriť kostru plánu, ako naplánovať samotný projekt a ako pomerne presne určiť časový harmonogram projektu.

Exaktné metódy v plánovaní

Tento prístup hľadí na problém plánovanie ako na technický problém. „Exaktné metódy používané v exaktnom plánovaní vychádzajú z poznatkov klasickej matematiky a štatistiky a tiež z poznatkov systémovej a operačnej analýzy“ [4]. Tento prístup sa však zameriava na časové projektové plánovanie a jeho výstupom je časový postup prác pri realizácii projektu s ohľadom na disponibilné zdroje, kapacity a náklady.

Tieto metódy však vychádzajú z predpokladu, že pre každú činnosť vieme odhadnúť, koľko bude trvať. Trvanie jednotlivých činností nie je však možné v mnohých prípadoch dostatočne spoľahlivo určiť. Komplikácie nastávajú predovšetkým vo výskumných projektoch, ktoré sú z viacerých hľadísk vždy ojedinelé, a teda s ich realizovaním nemá riešiteľský tím dostatočné skúsenosti. Takáto situácia môže nastať aj v rámci tímového projektu, lebo niektoré zadania majú výskumný charakter, v mnohých prípadoch členovia tímu – študenti nemajú predchádzajúce skúsenosti s podobným typom zadania a nedokážu dostatočne presne odhadnúť trvanie jednotlivých aktivít. A keďže ide často o novátorské projekty, počas riešenia sa často môžu objaviť nečakané problémy.

Exaktné metódy plánovania sú teda vhodné pre projekty, ktoré pozostávajú z mnohých rutinných procesov a aktivít.

Architektúra organizácie a jej vplyv na tvorbu plánu

V predošlom texte som kládol dôraz na technický pohľad na plán a jeho tvorbu. Nepadla totiž zmienka o ľuďoch, ktorí jednotlivé procesy vykonávajú. V mojom ponímaní, technický prístup chápe človeka ako stroj, ktorý sa drží stanovených postupov a vykonáva jednotlivé kroky. Avšak môže sa stať, a v realite často stáva, že projekt zlyhá, alebo sa omešká, zavinením ľudského faktora.

V tomto dokumente som sa venoval niekoľkým prístupom v rámci plánovania. Jednotlivé prístupy som spomenul stručne, lebo sú si v istom zmysle podobné a pozerajú na jeden a ten istý problém trochu odlišným spôsobom. V každom prípade sa vychádza zo špecifikácie požiadaviek a ako výsledok sa chápe softvérový produkt. Preto by som sa chcel v ďalšom texte sústrediť na personálny a osobnostný aspekt plánovania vývoja softvéru.

Za vhodné riešenie spejúce k presnému a úspešnému plánovaniu by som považoval snahu o elimináciu možných rizík, ktoré sa môžu objaviť počas vývoja. Toto riešenie by som nazval prevenciou. V predchádzajúcich kapitolách som sa

venoval opisu jednotlivých krokov a procesov týkajúcich sa tvorby plánu. Avšak v mnohých tímoch sa zabúda na ľudský faktor. Teda mnoho projektov stroskotá alebo sa oneskorí z dôvodu straty kľúčových členov, z dôvodu nedorozumení medzi jednotlivými členmi v rámci tímu alebo z dôvodu nedorozumení medzi tímami v rámci väčšieho projektu. Tieto faktory predstavujú významné riziká, avšak vhodnými opatreniami je možné týmto rizikám predchádzať.

V predošlých kapitolách bol kladený dôraz predovšetkým na riešený problém a menej sa pozeralo na riešiteľov. I napriek dokonalému plánu a precíznemu definovaniu jednotlivých procesov, sa môže napríklad stať, že členovia tímu stratia o projekt záujem alebo dôjde k mnohých neočakávaným prekvapeniam z dôvodu nedostatku komunikácie medzi tímom alebo samotnými členmi tímov.

V nasledujúcej časti eseje opíšem niekoľko vzorov, ktoré môžu zlepšiť kultúru správania v rámci organizácie alebo tímu. Aplikovaním týchto vzorov v reálnom prostredí bude možné ľahšie dosiahnuť stanovené ciele projektu včas, lebo dané vzory hľadia na ľudský faktor v rámci plánovania projektov. Budem sa venovať trom vzorom: neformálna sieť, rotácia a vyhýbanie sa prekvapeniam [8].

Neformálna sieť

Tento organizačný vzor kladie dôraz na komunikáciu v rámci projektu. Väčšinou je projekt rozdelený do modulov a vývojári preberajú zodpovednosť za jednotlivé moduly. Ako bolo spomenuté už v predchádzajúcich častiach eseje, vývojár si sám určí, koľko času bude potrebovať na podrobný návrh, implementáciu a testovanie určitého modulu. Avšak medzi jednotlivými modulmi existujú závislosti a funkcionalita niektorých modulov sa môže prekrývať, a tak programátor stojí pred rozhodnutím, či sa spoľahne sám na seba, vytvorí si všetky časti modulu sám, a tak vznikne duplicita v rámci projektu alebo použije časť práce, ktorá už bola vykonaná jeho kolegom. Druhé riešenie je efektívnejšie, avšak vyžaduje dostatočnú dokumentáciu k modulu, ktorý chce programátor použiť a významne by mu pomohla aj určitá pomoc od programátora zodpovedného za daný modul. Preto sa tento organizačný vzor zameriava na efektívnu komunikáciu medzi vlastníkmi modulov. V [8] je možné sa dočítať, že ako vhodný nástroj je možné použiť napr. fórum, ale na druhej strane je tiež potrebná podpora zo strany manažérov projektu, ktorí budú kooperatívne správanie vývojárov podporovať.

S týmto problémom sa je možné bežne stretnúť aj v tímovom projekte i napriek tomu, že tím nemá veľký počet členov. Často sa stane, že niektoré časti sú vyvíjané viacerými členmi naraz a to len z dôvodu nedostatočnej ochoty členov informovať tím o práci na svojom module a o jeho funkcionalite.

Rotácia

Autori v [8] uvádzajú ďalší organizačný vzor, ktorý je možné spomenúť v kontexte plánovania, a tým je rotácia kľúčových programátorov, teda tých, ktorí majú úplnú znalosť modulu, za ktorý sú zodpovední. Strata takých členov v rámci projektu by mohla významne ohroziť plán projektu. Odchod kľúčových členov je skutočnosť,

s ktorou sa pri plánovaní bežne nepočíta a preto je vhodné ošetriť ju na úrovni organizácie tímu a to tak, že programátori zodpovední za moduly budú postupne rotovať a pri rotácii predajú svoje vedomosti ďalej vývojárov, ktorí budú následne za moduly zodpovední.

Aj v tímovom projekte môže nastať situácia, že by člen tímu stratil záujem o predmet alebo o projekt a jednoducho by sa na práci tímu prestal podieľať. Rotáciou tento problém v rámci tímového projektu však nie je možné ošetriť. Preto je vhodnejšie budovať neformálnu sieť v rámci tímu a tak predchádzať strate motivácie jednotlivých členov.

Vyhýbanie sa prekvapeniam

Riešitelia majú často tendenciu neinformovať ostatných členov tímu v prípade, že sa dostali k časovým sklzom pri práci na pridelenom module. Na prvý pohľad to teda môže vyzerať, že projekt postupuje podľa harmonogramu. Keď programátor už nie je schopný odovzdať modul v stanovenom termíne, často si prácu zjednoduší vypustením určitých častí modulu, zjednodušením implementácie rozhraní a podobne. Problémy sa však objavia až pri integrácii, kedy dôjde k významným oneskoreniam z dôvodu opravy chýb, ktoré vznikli snahou o urýchlenie procesu vývoja modulu. Predísť nemilým prekvapeniam v čase integrácie je možné tak, že členovia jednotlivých tímov budú pravidelne informovať ostatných o prípadných zmenách a rizikách počas vývoja.

Aplikácia organizačných vzorov

Technický pohľad na plánovanie umožňuje často len identifikovať oneskorenia v rámci harmonogramu a následne aktualizovať plán podľa potreby. Organizačný vzor sa snaží vylúčiť neočakávané prekvapenia.

V predchádzajúcom texte boli načrtnuté niektoré organizačné vzory, ktoré by mohli byť vhodne použité v rámci zlepšenia plánovania v tímovom projekte. V literatúre sa uvádza veľké množstvo rôznych organizačných vzorov. Mojim cieľom nebolo detailne popísať jednotlivé vzory, chcel som poukázať na fakt, že technickým prístupom sa ku dokonalému plánu nie je možné dopracovať.

„Spomenuté organizačné vzory predstavujú neformálne postupy pre dosiahnutie efektívneho vývoja, sú chápané ako doplnok ku formálnym metódam“ [8]. Teda postupy, ktoré som spomenul v jednotlivých častiach eseje sú vhodné a použiteľné, avšak organizačné vzory môžu významným spôsobom dopomôcť pretaviť formálne a technické metódy do praxe.

Aplikácia v tímovom projekte

Z dôvodu zloženia nášho tímu mal náš projektový plán prevažne technický charakter. To znamená, že išlo o popis jednotlivých krokov, ktoré nás majú doviesť k riešeniu stanoveného problému. Ako problematické sa javilo samotné zloženie tímu vzhľadom na tému nášho projektu - robocup. Väčšia časť tímu sa problematike robotického futbalu venovala už počas štúdia v rámci niektorých predmetov. Teda išlo o významnú

nekonzistenciu v skúsenostiach a vedomostiach členov tímu. Badať to bolo možné napríklad v komunikácii medzi jednotlivými členmi. Tento problém a jeho riešenie je všeobecne načrtnutý v predchádzajúcej kapitole.

Na základe špecifikácie sme vytvorili vhodnú architektúru. Už zo samotnej architektúry vyplývala určitá postupnosť krokov, ktorá sme mali uskutočniť. Na základe skúseností z rôznych zadaní počas štúdia mali členovia nášho tímu predstavu o základných procesoch. Problém však často nastal v komunikácii. Preto za dôležitú súčasť plánovania považujem jeho sociálny rozmer.

Záver

Plánovanie je nesporne dôležitá aktivita v rámci procesu vývoja softvéru. Zároveň je však dôležité podotknúť, že nie je vhodné preceniť technický význam plánovania – teda odpovedať na otázku kedy, čo a ako. Najväčšou hodnotou v organizácii alebo v tíme sú totiž ľudia. Je teda vhodné vytvoriť členom predpoklady, aby mohli jednotlivé plány plniť. Organizačné vzory môžu vhodne prispieť k budovaniu robustných a flexibilných tímov, ktoré budú schopné plniť pridelené úlohy načas a v požadovanej kvalite.

Použitá literatúra

1. Johnson, D.L., Brodman, J.G.: Applying CMM project planning practices to diverse environments, IEEE, 2000
2. McConnell S.: The Nine Deadly sins of Project Planning, IEEE Computer Society Press, Los Alamitos (2001)
3. Tsoi H.L.: A Framework for Management Software Project Development, ACM Press, New York (1999)
4. Chovanova H.: Možnosti využitia exaktných metód v projektovom plánovaní
5. Paulish D.J., Nord R.L., Soni D.: Experience with architecture-centered software project planning, AMC Press, New York (1996)
6. Griswold G.G: Just-in-Time Architecture: Planning Software in an Uncertain World, New York (1996)
7. Retting M., Simons G.: A Project Planning and development process from small teams, ACM Press, New York (1996)
8. Dikel D., Hermansen Ch., Kane D., Malveaux R.: Organizational Patterns for Software Architecture, 1997

Annotation

Perfect Plan

Planning plays very important role in software development process. Development of each software product is unique process; that is why it is not enough to define the plan just as a sequence of steps. In the beginning phases of the project it is not possible to define all the steps due complexity of the software, moreover the world of software engineering is very dynamic and the plan should be flexible enough to cover all the changes. Essay focuses on plan definition on enterprise architecture level. Essay does not want to be a set of instructions, how to develop a perfect plan. Essay wants to present how to define the organization structure and its processes to fulfill the project aims on time and in sufficient quality.