POMOCNÉ KOLIESKA ROZHODOVANIA A RIADENIA

Kto chce vedieť, musí vidieť

Tomáš Uherčík

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

Abstrakt. *Vývoj softvéru je veľmi zložitá a rôznorodá problematika. Na jednom projekte zvyčajne pracuje viacero ľudí. Každý z nich má na starosti určitú oblasť, v ktorej je expertom. Aby bol tento proces efektívny, musí byť dobre riadený. Čo je však potrebné na úspešný manažment v procese vývoja softvéru? Dôležité je aby manažéri mali dostatočné podklady na konanie svojich rozhodnutí. Nie je však možné, aby rozumeli všetkým činnostiam, ktoré vykonávajú ich podriadení. Na riešenie tohto problému sa používajú rôzne metódy a nástroje, ktoré majú priblížiť prácu programátorov a analytikov ich nadriadeným. Tento jav platí aj z opačnej strany pri definícii požiadaviek. Existujú aj iné oblasti v manažmente vývoja softvéru, kde sa dajú použiť podporné prostriedky? V tejto eseji pojednávam o oblastiach aplikácie podporných prostriedkov v softvérovom inžinierstve. Zameriavam sa na ich využitie v oblasti vizualizácie softvéru a definície požiadaviek*

Kľúčové slová: *podporné prostriedky, nástroje, vizualizácia softvéru*

Úvod

Softvérové projekty sú čoraz komplikovanejšie a zložitejšie. Je potrebné, aby sa na nich podieľali veľké tímy ľudí. Pokiaľ sú tieto tímy dobre organizované dokážu vytvoriť viac, ako by dokázali jednotliví členovia tímov osobitne. Čím je však tím väčší, tým zložitejšie je efektívne riadenie. Avšak táto cesta je jedinou možnosťou úspešnej realizácie väčších projektov. Ďalším dôležitým aspektom je, že každý z členov tímu je odborníkom na svoju oblasť. Nie je potrebné, aby rozumel všetkému, čo obsahuje celý systém. Má na zodpovednosť časť, ktorú mu pridelí vedenie. Je však nevyhnutné, aby nadriadení poznali

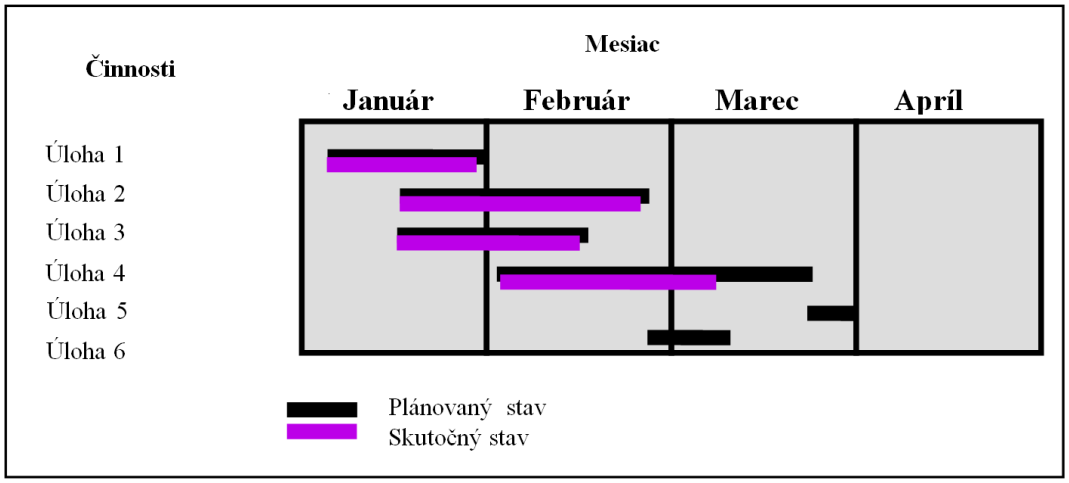
celý systém? Po krátkom zamyslení, možno aj bez neho, je hneď každému zrejmé, že nikto nemôže poznať každý detail ktoréhokoľvek z väčších softvérových projektov. Manažéri však musia chápať koncepciu časti systému, ktorej vývoj majú riadiť. Dalo by sa povedať, že čím vyššie v hierarchii riadenia je manažér, tým je väčšia časť systému, za ktorú zodpovedá a tým vyššia je úroveň abstrakcie, s ktorou sa pozerá na systém. Dosiahnuť aby každý z manažérov videl do vývoja systému dostatočne na to, aby dokázal robiť úspešné rozhodnutia nie je triviálna záležitosť. Pri veľkosti dnešných softvérových projektov si to asi ani len nevieme predstaviť bez použitia prostriedkov, ktoré by nám tento proces uľahčili. Pod týmito prostriedkami môžeme rozumieť všetko, čo zlepší riadenie. Môžu to byť rôzne metodiky, postupy a schémy, ktoré zjednodušujú a zdokonaľujú manažment, alebo sú to priamo konkrétne softvérové podporné nástroje plniace túto úlohu. Dnešný vývoj je veľmi komplexnou záležitosťou a teda pri rôznorodých procesoch z tejto oblasti sú vhodné nápomocné nástroje.

Kde použiť podporné prostriedky?

Základným procesom, v ktorom je vhodné použitie podporných prostriedkov je riadenie. Má mnoho spoločného so všeobecnými postupmi používanými aj pri iných ako softvérových projektoch. Pri riadení akéhokoľvek projektu je potrebné rozdelenie úloh a sledovanie priebehu plnenia jednotlivých úloh.

Medzi základné techniky používané pri manažmente projektu môžeme zaradiť napríklad[1]:

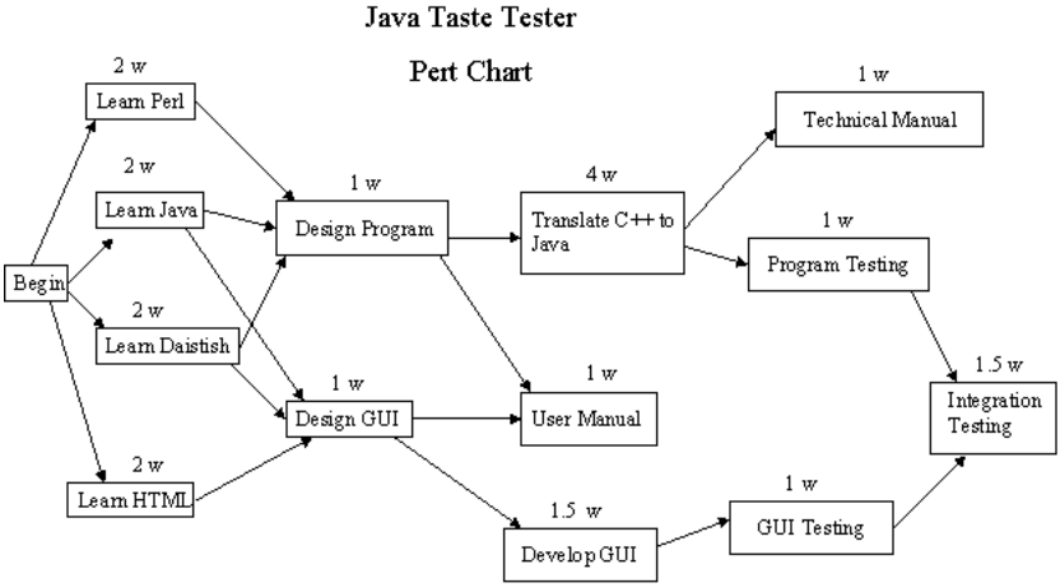
1. Ganttov graf – Je to základná technika na reprezentáciu fáz a aktivít projektovej štruktúry. Zvyčajne je definovaný typom stĺpcového grafu, ktorý znázorňuje priebeh jednotlivých úloh v projekte. Zobrazuje začiatočný aj konečný čas jednotlivých aktivít. Jeho použitie umožňuje názornejšie sledovanie úloh.



Obr. 1. Ganttov diagram

2. PERT diagram – Slúži na zobrazenie projektu sieťovým diagramom, pozostávajúcim z viacerých uzlov, ktoré reprezentujú míľniky a udalosti. Uzly sú pospájané vektormi, ktoré predstavujú úlohy v projekte. Smer šípok vektorov indikuje následnosť úloh. Použitie je vhodné, ak je potrebné[2]:

   - určiť najkratší čas, za ktorý je možné dokončiť projekt
   - určiť kritickú cestu, teda určiť úlohy, ktoré sú kritické z hľadiska včasného ukončenia projektu
   - určiť časové úseky, kde sa má súbežne vykonávať viac úloh



Obr. 2. PERT diagram[3]

3. CPM – Metóda kritickej cesty. Slúži na hľadanie postupnosti vykonania jednotlivých úloh tak, aby bol projekt, čo najskôr ukončený. Na hľadanie najkratšej cesty je nápomocný PERT diagram.

Napriek tomu, že riadenie softvérových projektov má veľa spoločného s riadením projektov z iných oblastí, predsa len ho môžeme považovať za niečo trochu odlišné, možno viac rizikové. Softvérové projekty sú náročné na manažment a veľa z nich končí neúspechom[4]. Preto je ich manažment veľmi netriviálnou oblasťou. Veď o čo je ľahšie odhadnúť za aký čas sa podarí postaviť budovu, keď približne vieme koľko trvá

postavenie podobnej? Aký je ale podobne veľký softvérový systém? Ako dokážeme odhadnúť trvanie takejto kreatívnej činnosti? V čom sa to líši od ukladania tehál?

Po zvážení špecifických čŕt softvérových projektov, môžeme identifikovať tieto základné oblasti, v ktorých je vhodné použitie podporných prostriedkov[5]:

1. **Plánovanie úloh** – Priradenie začiatočného a konečného času úlohám. Vďaka tomu môžu manažéri softvérových projektov sledovať dôležité míľniky a zaznamenať, kto je za čo zodpovedný. Získame tým určitý prehľad o tom ako sa jednotlivé úlohy spĺňajú v súvislosti s tým, kto na nich pracuje.
2. **Manažment zdrojov** – Umožňuje manažérom organizovať a sledovať detaily požiadaviek, aby sa uistili, že sú pre projekt vyčlenené potrebné zdroje. Patrí sem aj prideľovanie a menenie informačných atribútov, akými sú priority a stav úloh.
3. **Kolaborácia** – Umožnenie zdieľania znalostí a dobrých praktík. Zahŕňa protokoly viditeľné pre všetkých členov tímu. Je potrebné, aby bola vysoká úroveň formálnej komunikácie v rámci vývojového tímu. Bez nej by sa mohli členovia tímu vyhovárať, že o niečom nevedeli, alebo im to nebolo pridelené.
4. **Sledovanie času** – Je potrebné sledovanie času vzhľadom na pomer splnených a nesplnených úloh, aby bolo možné vyhodnotiť, ako rýchlo sa projekt vyvíja. Dôležité je porovnanie reálneho času s plánom.
5. **Odhadovanie** – Umožňuje projektovým manažérom generovať a overovať veľkosť približného potrebného úsilia. Vyhodnocuje projektový plán pre rôznorodé druhy projektov.
6. **Hodnotenie rizík** – Pomáha manažérom identifikovať riziká projektu. Tiež je nápomocné pri popisovaní a ohodnotení rôznych rizikových faktorov. Jedna z najdôležitejších a zároveň najťažšie dosiahnuteľných úloh vzhľadom na špecifiká softvérových projektov.
7. **Manažment zmien** – Umožňuje sledovanie vykonaných zmien a prípadný návrat na predošlé verzie projektu. Ak nastanú zmeny, ktoré systém dostanú do horšieho stavu, je možné ich vrátiť späť.
8. **Manažment procesu** – Slúži na vytvorenie konzistentného a štandardizovaného toku. Pre každý systém je charakteristický určitý proces následných aktivít, ktorý treba namodelovať.
9. **Manažment portfólia** – Pomáha manažérom spravovať súvisiace projekty. Pre manažérov je výhodné, ak vedia, že jeden projekt sa do určitej miery zhoduje s druhým. Vďaka tomu nemusia hľadať riešenia, ktoré už boli nájdené.

Aj keď existuje veľmi široká škála prostriedkov používaných v softvérovom inžinierstve, väčšina z nich sa zaoberá orientáciou v tom, čo sa deje v čiernej skrinke procesu vývoja softvéru alebo zvýšením úrovne komunikácie medzi členmi tímu. Neprehľadnosť teda začína už na najvyššej vrstve riadenia. Sú však tieto prostriedky všetkým čo potrebujeme? Ako sprostredkovať informácie z nižších vrstiev, ktoré sú potrebné pre rozhodnutia v riadení?

Každý člen tímu, či už je to programátor, analytik alebo manažér, by mal rozumieť oblasti, ktorú vyvíja alebo riadi. Každý z nich na úrovni abstrakcie príslušnej postaveniu, ktoré v rámci projektu zaujíma. U analytikov a programátorov je predpokladom, že

poznajú notácie na zobrazenie softvéru, aby vedeli, na čom majú pracovať. Ako však pomôcť manažérom a zákazníkom chápať softvér?

Problémy s vizualizáciou softvéru

Pri návrhu softvéru nie je efektívne, aby analytik robil rozhodnutia na základe rozsiahlych postupov a pravidiel, lebo nech by boli pravidlá akokoľvek rozsiahle, pravdepodobne nikdy neobsiahnu všetky rôzne situácie, ktoré môžu nastať. Je nutná konzultácia s manažérmi, ktorí sú kompetentní robiť strategické rozhodnutia. Nastáva však problém, ako znázorniť ľudom, ktorí nie sú z IT, softvér tak, aby ho pochopili dostatočne na to, aby boli schopní spraviť potrebné rozhodnutia. Nakoľko jednou z vlastností softvéru je jeho neviditeľnosť a teda neexistuje jeho komplexná reprezentácia, je problémom ho zobraziť tak, aby bol dostatočne pochopiteľný.

Jednou z možností je naučiť manažérov čítať niektoré diagramy používané analytikmi. Je to síce riešenie, ktoré prináša náklady spojené s ich vzdelávaním, umožní im však lepšie porozumenie softvéru. Nevýhodou je, že to nie je možné naučiť všetkých manažérov a taktiež je úplne nemožné využitie tejto metódy na strane zákazníka. Riešenie je možné vidieť v kompromise: naučiť nižšie postavených manažérov rozumieť diagramom, s ktorými sa stretávajú, čím získajú lepšiu predstavu o vyvíjanom softvéri. Nakoľko u nich sú predpokladané lepšie komunikatívne schopnosti a manažérske znalosti, dokážu svoju predstavu predniesť vyššiemu vedeniu alebo zákazníkovi, so zameraním na aspekty, v ktorých je potrebné prijatie rozhodnutia.

Ďalšou možnosťou je použiť softvérové nástroje na vizualizáciu softvéru. Tieto nástroje by mali podporovať abstrakčné mechanizmy, rôzne manipulačné techniky a trojdimenzionálne textúry[6]. Mali by dokázať analyzovať komplexné systémy a byť nápomocné aj pri údržbe a spätnom inžinierstve. Sila vizualizácie je odvodená od bohatosti sémantiky, jednoduchosti a stupňa abstrakcie[6]. Ideálnym cieľom je vyvinúť jazyk, obsahujúci iba niekoľko metafor a konštrukcií, ktorý by dokázal reprezentovať rôzne variácie elementov bez vzniku nejednoznačnosti alebo straty významu[6]. Metafory by mali byť jednoduché a priamočiaro príbuzné cieľu. Reprezentáciu systému môžeme vyjadriť napríklad zdrojovým kódom, tabuľkami, diagramami, grafmi alebo vizuálnymi metaforami, akými sú ikony, obrázky, príznačné slová a pod. Taktiež je potrebné zvážiť atribúty, akými sú interaktívne, statické, dynamické náhľady, rôzne formy zobrazenia, rôzne stupne abstrakcie a pod.[6]. Sedem úrovní, ktoré by mala aplikácia slúžiaca na vizualizáciu obsahovať[6]:

1. **Prehľad** – Zobrazenie celkového náhľadu na reprezentáciu dát. Je to zvyčajne zložitý problém pri vizualizácii štruktúrovaných informácií vo veľkých systémoch. Táto reprezentácia je pri väčších systémoch len veľmi ťažko čitateľná zo zdrojového kódu. Táto funkcia je veľmi dôležitá na získanie celkového prehľadu o systéme.
2. **Priblíženie a vzdialenie** – Priblíženie zaujímavých častí. Pri priblížení a vzdialení je dôležité zachovanie globálneho kontextu. Dosahuje sa to rôznymi stupňami abstrakcie. Ak chceme rozumieť systému, mali by sme mať určitý prehľad o jeho všetkých súčastiach, aby sme vedeli zvážiť súvislosti, ktoré

v systéme vznikajú. Na druhej strane je dobré, ak sa dokáže zobrazenie zamerať len na časť, ktorá nás zaujíma. Zrejmé je využitie vo veľmi veľkých systémoch, kde aj relatívne malá časť môže mať v skutočnosti obrovský rozsah.

3. **Filter** – Možnosť odfiltrovania nezaujímavých častí. Časti, ktoré pre nás nie sú zaujímavé iba odvracajú našu pozornosť od podstatných informácií. Filtrácia má svoje neprehliadnuteľné výhody aj v eliminácii nedostatkov zobrazovacích možností zariadení, ktoré reprodukujú obraz. Odstránenie nezaujímavých častí však vyvolá narušenie celkového kontextu, preto je dôležitá podpora nejakého spôsobu abstrakcie odstránených častí, aby nedochádzalo k strate dôležitých súvislostí.
4. **Detaily na požiadanie** – Výber častí, ktoré sú zaujímavé a zobrazenie potrebných detailov. Pre vývojárov je často potrebné zobrazenie aj tých najpodrobnejších detailov systému.
5. **Vzťahy** – Zobrazenie vzťahov, medzi jednotlivými časťami. Pre hierarchické dátové štruktúry je potrebné zobrazenie vzťahu rodič-dieťa. Toto je jedna z najdôležitejších vlastnostní vizualizačných systémov, pretože popisuje vzájomné prepojenia. Vzťahy medzi jednotlivými časťami sú významné, lebo popisujú systém ako celok.
6. **História** – Je užitočné si pamätať históriu akcií, aby bolo možné vykonať spätnú akciu.
7. **Extrakcia** – Umožnenie extrakcie vnorených dátových štruktúr a použitia dopytov s parametrami. Táto možnosť predstavuje náročne pochopiteľnejšie rozhranie, ktoré by však umožňovalo oveľa flexibilnejšie získavanie zaujímavých častí.

Treťou identifikovanou možnosťou je použitie agilných metód vývoja. Tieto metódy plynú zo skutočnosti, že požiadavky bývajú často nejasné a často sa menia. Presné nasledovanie plánu už dlhšie nie je primárnym cieľom, prvoradou úlohou je uspokojenie zákazníka v čase odovzdania riešenia, nielen na začiatku pri schválení požiadaviek[7]. Základným princípom je čo najrýchlejšie dodanie prvého riešenia, ktoré môžeme následne konzultovať so zákazníkom. Fungujúci kód je niečo, čo majú zákazníci priamo pred sebou a teda vidia reálny systém. Je to asi najpriamejšia možnosť zobrazenia. Odpadajú rôzne metafory a abstrakcie, zákazník vidí to, čo systém v danom okamihu obsahuje. Vďaka interakcii medzi jednotlivcami, ktorí majú dostatočné informácie, ľahko dosiahneme, že sa proces vykonania potrebnej zmeny skráti. Používanie už fungujúceho čiastočného riešenia nám umožňuje odmerať, ako rýchlo v skutočnosti dosahujeme výsledky a tiež nám to poskytuje rýchlu spätnú väzbu[7]. Časté interakcie medzi jednotlivými členmi tímu kompenzujú minimum dokumentácie. Spolupráca so zákazníkom znamená, že všetci hráči (sponzor, zákazník, používateľ a vývojári) sú v jednom tíme[7]. Spojenie ľudí s bohatými skúsenosťami s expertmi umožňuje kombinovanej skupine rýchlo meniť smer a vyprodukovať lepšie výsledky s menšími nákladmi[7]. Väčšina metodológií poskytuje kompletné pravidlá na všetky možné situácie, ktoré môžu nastať. Agilné metódy na druhej strane prinášajú generujúce pravidlá. Teda tím závisí od kreativity jeho členov[7]. Pri použití kompletných pravidiel závisí úspech od niekoho, kto ich všeobecne definuje, s tým, že pravdepodobne nemôže zvažovať všetky špecifiká situácií, ktoré môžu nastať.

Kreativita, nie rozsiahle pravidlá, je jedinou cestou, ako manažovať problémy komplexného vývoja softvéru[7].

Definícia požiadaviek

Ďalším problémom v softvérovom inžinierstve je, že neexistuje ideálny spôsob definovania požiadaviek. Požiadavky sú zväčša definované zákazníkom, respektíve vyšším manažmentom. Táto skupina ľudí neovláda nástroje na ich definíciu, ktorá by bola zrozumiteľná pre programátorov. Prirodzenou cestou pre nich je ich textová definícia. Každému človeku je prirodzené definovať to, čo požaduje formou svojho prirodzeného jazyka, ktorým primárne komunikuje s okolím. Možnosti vyjadrenia týmto spôsobom sú prakticky neohraničené. Táto reprezentácia je však problematická z viacerých hľadísk. Už samotná neobmedzenosť prináša príliš veľkú voľnosť, ktorá môže v konečnom dôsledku spôsobiť nejednoznačnosti v pochopení významu. Pre takéto účely je teda potrebná jednotná dohoda o tom, ako sa majú problémy vyjadrovať. Ďalšou neodmysliteľnou nevýhodou je značná neprehľadnosť. Je možné si predstaviť systém definovaný na desiatich stranách čistého textu? Aj keď by to bolo možné, aké úsilie by bolo potrebné, aby im porozumel iný človek?

Riešením je definícia požiadaviek použitím dohodnutých notácií, ktoré sú jednoznačné a prehľadné. Problém však nastáva s ich tvorbou. Ich tvorba nie je človeku priamo prirodzená a ľuďom, ktorí nie sú priamo z IT sektora môže robiť problémy. Je preto potrebné, aby ich definovali IT odborníci. Ak však majú konzultovať priamo s manažérmi, môže byť tento proces zdĺhavý, nákladný a nejednoznačný.

Ako skĺbiť prirodzenosť textovej definície požiadaviek s vytvorením diagramov v zaužívaných notáciách? Ideálnym riešením by bola transformácia textu do požiadaviek v grafických notáciách. Vývojom nástrojov, ktoré pomocou spracovania prirodzeného neštruktúrovaného jazyka vytvárajú požiadavky sa zaoberalo viacero prác. Jednou z nich je [8]. Táto štúdia sa zaoberá vytváraním diagramu tried z textovo definovaných požiadaviek. Autori uvádzajú, že je to veľmi ťažká úloha, čo nemožno v žiadnom prípade spochybniť. Ich nástroj používa spracovanie prirodzeného jazyka na získanie znalostí z požiadaviek. Koncepty a vzťahy sú reprezentované ontológiou, ktorá je zdrojom doménových znalostí. Použitím pravidiel pre extrakciu tried sa vytvorí diagram tried, ktorý obsahuje vzťahy generalizácie, asociácie, kompozície, agregácie a závislosti. Nepodporuje však kvantifikátory vzťahov. Ich práca je obdivuhodná a zároveň môže byť veľmi užitočná. Je však možné týmto spôsobom generovať kompletné diagramy? Aj za predpokladu, že by tento spôsob fungoval perfektne, je zrejmé, že na vytvorenie kompletného diagramu potrebujeme kompletné požiadavky. Ako však donútiť zainteresovaných písať požiadavky, ktoré obsahujú všetky potrebné informácie, bez definície štruktúry požiadaviek?

Ideálne by bolo použitie textu vhodného na následné strojové spracovanie. Bol by ním štruktúrovaný text, ktorý by obsahoval všetky potrebné informácie. Týmto spôsobom by sa dalo jednoducho získať všetky potrebné informácie. Konkrétne by sme to mohli dosiahnuť napríklad vytvorením rozhrania, ktoré by obsahovalo preddefinované časti, polia, ktoré poskytnú používateľovi výber slova z ponuky slov a samozrejme polia do ktorých bude môcť písať ľubovoľné slová. Takýmto spôsobom by sa teoreticky dalo

dosiahnuť vytvorenie použiteľných diagramov bez toho, aby ich tvorca vedel, čo je potrebné definovať na ich tvorbu. Výsledkom by mali byť požiadavky definované zrozumiteľným prirodzeným jazykom, ktoré je následne možné transformovať do grafického modelu.

Záver

V eseji pojednávam o tom, kde všade v oblasti vývoja softvérových projektov je vhodné použitie podporných prostriedkov. Zameriavam sa na javy v softvérovom inžinierstve, ktoré nie sú na prvý pohľad viditeľné a sú dôležité pri riadení a rozhodovaní. Sú to napríklad procesy v manažmente, akými sú sledovanie úloh a ich plnenie ľuďmi, ktorí sú za ne zodpovední. Najneviditeľnejším v tomto procese je však softvér samotný. Preto zvažujem, čo je potrebné na to, aby bol softvér aspoň o niečo viditeľnejší pre jednotlivých členov tímu v softvérovom projekte. Snažím sa o identifikáciu možných riešení vizualizácie softvéru, ktoré by uľahčili rozhodovanie a definíciu požiadaviek.

Použitá literatúra

1. Murphy, M.G.: Teaching software project management : a response - interaction approach. *In: Journal of Systems and Software*, Vol. 49 (1999) 1997-2000.
2. Manažment promotion [online] [cit. 2010-10-15]. Dostupné na internete: <www.valdner.com/school_public/FMK%20UCM%20TT/5roc%20%20Manazment%20promotion/prezenacia.ppt >
3. Java Taste Tester. PERT diagram. [*online] [cit. 2010-10-15].* Dostupné na internete: <http://www.cs.unc.edu/~stotts/danish/web/gifs/pert.gif>
4. Keil, M., Cule, P.E., Lyytinen, K., Schmidt, R.C.: A Framework for Idenifing Software Project Risks. *In: Communications of the ACM*, Vol. 41, (1998).
5. Ahmad, N., Laplante,P.: Software Project Management Tools: Making a Practical Decision Using AHP. *In: Annual IEEE/NASA Software Engineering Workshop*, Vol. 30, (2006) 76-84.
6. Marcus, A., Feng, L., Maletic, J.I.: 3D representations for software visualization *In Proceedings of the 2003 ACM symposium on Software visualization - SoftVis '03*, (2003), 27.
7. Highsmith, J., Cockburn, A.: Agile Software Development: The Business of Innovation. *In: Computer,* Vol. 34,(2001) 120-127.
8. Ibrahim, M., Ahmad, R.: Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques. *In: 2010 Second International Conference on Computer Research and Development*, (2010) 200-204.

Annotation

Auxiliary wheels of decision making and management

Software development is very complex and diverse field. There are usually a lot of people, working on one project. Everyone is expert in his area. To make this process effective, it has to be well managed. What is necessary for the successful management of the software development process? It is important for managers to have sufficient knowledge to make their decisions. However, it is not possible to understand all the activities carried out by their subordinates. To solve this problem, there are different methods and tools to make the work of programmers and analysts visible to their superiors. This phenomenon also exists on the opposite side, in the area of requirements definition. Are there other areas in the management of software development, where can be used supporting tools? In this essay I discuss the areas of application support tools in software engineering. I focus on their use in the area of visualization of software and definition of requirements.