SOWA

Etika softvérového inžiniera a softvérové procesy

MSI
KIVT STU
Bratislava

Etika softvérového inžiniera a softvérové procesy

Tím SOWA

Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky
2000

Vedúci autorského tímu:
Bc. Viktor Zeman

Autori
Bc. Ján Glončák, Bc. Róbert Kovaľ,
Bc. Kamil Krnáč, Bc. Richard Koncz,
Bc. Michal Kucbel, Bc. Branislav Lehotský,
Bc. Peter Liczki, Bc.Richard Mátefy,
Bc. Marián Varga, Bc. Viktor Zeman

Recenzovali:
Doc. Ing. Mária Bieliková, CSc.
Bc. Viktor Zeman
Bc. Marián Varga
Bc. Richard Koncz

Grafická úprava:
Bc. Viktor Zeman
Bc. Róbert Kováľ

Predslov

Práve ste si otvorili publikáciu, ktorá vznikla na predmete Manažment v softvérovom inžinierstve, čiže MSI. Autori sa v nej pokúšajú zachytiť svoje vedomosti a názory v dvoch na sebe nezávislých častiach. Prvá časť publikácie zachytáva názory autorov na etiku v softvérovom inžinierstve v podobe esejí reagujúcich na jednotlivé časti etického kódexu vytvoreného organizáciami IEEE Computer society a ACM. Výtvory v nej sa podobajú skôr umeleckým dielam ako odborným článkom. Preto na vykompenzovanie rovnováhy sme zaradili do publikácie aj odborné články na tému softvérové procesy.

Ako čitateľ musíte pochopiť, že spracovanie esejí v prvej časti je silne subjektívne. Aj keď je Etický kódex jednoznačný, názor naň môže mať každý človek iný. Keby ste náhodou s niektorými našimi názormi nesúhlasili, nevzdávajte sa, možno ihneď v nasledujúcej eseji sa vám podarí s nami stotožniť. Na oživenie celej prvej časti sme eseje vložili do príbehu, ktorý vám určite pomôže preklenúť aj menej zážívné pasáže...

Odborné články v druhej časti publikácie mali slúžiť na zachytenie najnovších trendov v softvérových procesoch. Keďže ide o veľmi rozsiahlu problematiku, po stručnom úvode sme sa zamerali len na niektoré špeciálne oblasti softvérových procesov.

Dúfame, že naplníme na nasledujúcich stranách vaše očakávania.

Obsah

Eseje – úvod

Problematika etiky softvérového inžiniera je v dnešnej dobe veľmi diskutovanou otázkou medzi predstaviteľmi odbornej verejnosti. Softvérové inžinierstvo prekonalo za posledných niekoľko rokov významné zmeny a preto vznikla medzi odborníkmi potreba vytvoriť samostatný etický kódex tejto profesie. Vzniklo mnoho etických kódexov, ktoré sa postupom času vyvíjajú a zdokonaľujú. V tejto časti publikácie sme sa snažili zaujať postoj ku jednotlivým bodom etického kódexu, ktorý vznikol ako výsledok spolupráce organizácií IEEE Computer society a ACM (Software engineering code of ethics and professional practice, verzia 5.2). Každá esej v tejto časti publikácie je reakciou na jeden bod tohto kódexu a jednotliví autori sa snažili vyjadriť svoj názor na etické otázky, ktoré kódex nastoľuje.

Aby sme publikáciu oživili, pokúsili sme sa o letmý pohľad do budúcnosti. Je zrejmé, že princípy etiky softvérového inžinierstva nie sú stále a nemenné, ale skôr naopak budú sa vyvíjať podľa aktuálnych potrieb profesie. Eseje sú spojené príbehom, ktorý sa odohráva v budúcnosti. Hlavný predstaviteľ tohto príbehu konfrontuje svoje skúsenosti (jeho skúsenosti sú z nášho pohľadu budúcnosťou) s informáciami, ktoré sa dozvedá pri čítaní esejí. Príbeh je vymyslený a je pokusom odhadnúť, ktorým smerom sa posunie chápanie etiky v budúcnosti.

Želáme Vám príjemné čítanie.

Chrómovo titánové výťahové dvere sa plynulo s kovovou eleganciou otvorili. Wesly Woodword vystúpil z výťahu a vydal sa do svojej kancelárie. Sadol si do pohodlnej ergonomickej stoličky a bez toho, aby to vyslovil, objednal si cez MindRead svoju obľúbenú írsku kávu (teda lepšie povedané jej vernú napodobeninu bez akýchkoľvek nežiaducich prísad ako kofeín a alkohol). Všetci jeho priatelia ho považovali za staromódneho, lebo hoci už niekoľko rokov vôbec nebolo potrebné odchádzať za pracovnými povinnosťami mimo svoj domov, Wesly sa tohto zvyku nechcel vzdať. Považoval to za svoju osobnú vec, vec štýlu alebo niečo podobné, a akýkoľvek technický vývoj a inovácie v tomto smere ignoroval. Bolo to podobné, ako keď v roku 2041 prestali vychádzať New York Times v tlačenej podobe na papieri. Nevedel sa zmieriť s myšlienkou, že sa bude musieť vzdať svojho ranného sobotného rituálu, keď sa na rozdiel od svojich priateľov a známych, ktorí čítali noviny len v sieti, alebo si ich dokonca nechali čítať počítačom, oddával celé sobotné dopoludnie potešeniu z čítania pravých, nefalšovaných papierových New York Times. Patril medzi iniciátorov protestov, ktoré nakoniec donútili redakciu NYT, aby predsa len pokračovala vo vydávaní novín i na papieri, aj keď len v minimálnych nákladoch. Wesly naozaj vyzeral na prvý pohľad staromódny. Nikto z jeho okolia nechápal prečo má doma namiesto bežnej plazmovej audio aparatúry staručký prehrávač CD platní. Málokto však vedel, že práve Wesly ešte v nedávnej minulosti patril ku špičkám v oblasti najnovších softvérových technológií. Už bolo len málo tých, ktorí si spomenuli na teórie ako WTS alebo metodológiu UFTCT. Tie už dávno zapadli prachom a boli vytlačené novšími a modernejšími. Wesly ešte pred piatimi rokmi pracoval na vývoji metodík a postupov pre softvérový návrh. Dnes si ešte na neho niekedy spomenie nejaká japonská univerzita a pozve ho prednášať o histórii softvéru, ale ku špičkám už rozhodne nepatrí. V jeho 35 rokoch však už patrí skôr medzi tých neperspektívnych a preto sa rozhodol, že začne pracovať pre NSHI (národný softvérový historický ústav) a bude sa skôr venovať minulosti vývoja softvéru. Ústav vznikol ako spoločný projekt firiem SunMicrosoft corporation a AmericanNet ako odložisko pre zastaralých programátorov, ktorí boli dosť vysoko v pozícii firiem na to, aby ich mohli poslať na ulicu. Wesly to samozrejme vedel, ale vôbec mu to neprekážalo. Celkom ho zaujímala história softvéru a pri jej štúdiu sa často veľa nasmial, keď čítal, aké primitívne problémy trápili programátorov v minulosti, no niekedy zas musel uznať , že mnohé myšlienky z minulosti majú nadčasovú platnosť.

Weslyho celodennou náplňou práce bolo študovanie historických materiálov a ich anotácia. Ústav sa snažil zo štúdia historických publikácií, časopisov, Web dokumentov objaviť v minulosti nejakú drobnosť, myšlienku prípadne celý postup, na ktorý sa zabudlo alebo bol vo svojom čase nepoužiteľný a dnes by sa realizovať dal. Občas sa im podaril celkom zaujímavý objav, ale naozaj to nebolo nič svetoborné. Niektorým odborníkom sa zdalo snaženie ústavu veľmi primitívne, lebo umelo-inteligentné automaty vedeli spracovať dokumenty 1000krát rýchlejšie. Ako však tvrdili odborníci z ústavu, automaty mnohokrát prehliadli skutočnosti, ktoré boli zrejmé len človeku.

Píp – píp – píp – píp – píp , Weslyho mozgový implantát WBC (Wave brain communicator v. 14) mu signalizoval, že dorazila objednaná káva. Podišiel ku otvoru v stene svojej kancelárie a vybral z neho kávu. Skôr než ju stihol dopiť, WBC sa znovu spojil s jeho mozgom a oznámil mu, že na sieti s ním chce hovoriť jeho šéf, Andrew Webber (ktorého všetci prezývali posmešne Loyd).

Weberova holografická projekcia sa objavila vedľa Weslyho a sadla si do kresla. Bol ako vždy vkusne oblečený. Wesly na ňom obdivoval, ako vyberavo si volí v komunikátore svoj holografický profil. On nastavoval ten svoj vždy na prednastavenú hodnotu a preto sa objavoval tam na druhej strane linky v jednoduchom, možno už trošku nemodernom obleku. Weber má fakt vkus, pomyslel si.

„Ahoj Wesly, ako ide deň ?“

„Nazdar hmmm Andrew.“ povedal Wesly a takmer mu vykĺzla prezývka Loyd.

„Mám sa fajn. A ty ?“

„Dík fajn, nemôžem sa sťažovať. Počuj Wesly, umelointeligentná analýza niektorých archívov nám vytypovala niekoľko dokumentov, ktoré pochádzajú z roku 2000. Sú to vraj riadne stariny, ale mohol by si sa na ne pozrieť. Čo ty na to ? Veď aj minule sme si mysleli, že z tej teórie nepravdepodobnosti nič nebude, a predsa bolo. Takže pozrieš sa na to ?“

Wesly len nemo prikývol, lebo vedel, že s Loydom sa nedalo vyjednávať. Bol zavalený hromadou inej roboty, ale aj napriek tomu rok 2000 znel lákavo. Pomocou WBC preložil všetky svoje naplánované úlohy na neskôr a v centrálnej databáze dokumentov ústavu si vyhľadal dokumenty, ktoré spomínal Loyd. S prekvapením zistil, že dokumentov nie je až tak veľa, ako si myslel. Vybral spomedzi článkov prvý, a premietol si ho na stenu. Telom sa mu prehnali príjemné zimomriavky. Už sa tešil na čítanie, lebo styk s históriou bol pre neho vždy potešením. Začítal sa do prvého článku...

Vzťah inžiniera k verejnosti a konflikt svedomia

Richard Mátéffy
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
mateffy@internet.sk

Abstrakt. Témou tejto eseje je vzťah softvérového inžiniera k verejnosti. Vychádza z *Kódexu etiky a profesionálnej praxe softvérového inžiniera*. Softvérový inžinier sa pri výkone svojho povolania môže dostať ľahko do situácie, keď jeho práca, či produkt jeho práce bude v rozpore s etikou a spoločenským dobrom. Rozriešenie takejto situácie nie je triviálne. Často závisí aj od interpretácie etiky inžinierom. Preto sa táto esej najprv snaží vymedziť pojem etika. Potom na príklade katastrofy raketoplánu Challenger ukázať príklad konfliktu svedomia, zovšeobecniť ho a ak nie nájsť všeobecné riešenie, tak aspoň zamyslieť sa nad problémom a uviesť nejaké princípy.

Etický kódex softvérového inžinierstva ACM a IEEE, 1. princíp: Verejnosť [1]

Softvéroví inžinieri majú konať v zhode s verejným záujmom. Konkrétne, softvéroví inžinieri majú, tak aby to bolo čo najvhodnejšie:

1.01. Akceptovať plnú zodpovednosť za vlastnú prácu.
1.02. Zosúladiť záujmy softvérového inžiniera, zamestnávateľa, klienta a používateľa s verejným dobrom.
1.03. Schváliť softvér, iba ak majú opodstatnenú vieru, že je bezpečný, spĺňa špecifikácie, prešiel primeranými testami a neznižuje kvalitu života, neuberá zo súkromia, ani nepoškodzuje životné prostredie.
1.04. Odhaliť príslušným osobám a autoritám každé skutočné alebo potenciálne nebezpečenstvo pre používateľa, verejnosť alebo životné prostredie, o ktorom odôvodnene veria, že je spojené so softvérom alebo so súvisiacimi dokumentmi.
1.05. Prispieť k diskusii o otázkach so závažným verejným dopadom spôsobených softvérom, jeho inštaláciou, údržbou, podporou alebo dokumentáciou.

1.06. Byť čestný a vyvarovať sa zavádzaniu vo všetkých vyhláseniach, obzvlášť vo verejných, zaoberajúcich sa softvérom alebo súvisiacimi dokumentmi, metódami a nástrojmi.
1.07. Zvážiť problémy fyzického handicapu, rozdelenia zdrojov, ekonomických znevýhodnení a iných faktorov, ktoré môžu znemožniť prístup k úžitku zo softvéru.
1.08. Mať odvahu dobrovoľne poskytnúť profesionálne zručnosti na dobré veci a na prispenie k verejnému vzdelaniu týkajúcemu sa disciplíny.

Vzťah inžiniera k verejnosti a konflikt záujmov

Niečo o etike

Každý človek sa snaží, alebo by sa mal snažiť správať správne, najmä čo sa týka vzťahu k jeho okoliu a k ostatným ľuďom. Inžinieri sa podieľajú na vývoji diel, produktov, ktoré môžu ovplyvniť celú spoločnosť, zvýšiť množstvo spoločenského dobra alebo zla. Preto je u nich o to naliehavejšie, aby sa správali a konali správne a morálne. V súčasnosti je na celom svete prudký rozvoj informačných technológií a s ním spojený rozmach softvéru. Softvér začína zasahovať na celom svete do života každého človeka a život bez softvéru a počítačov by sme si v súčasnom modernom svete nevedeli ani predstaviť. Keďže softvér ovláda veľkú časť aspektov nášho života, leží veľká osobná, morálna zodpovednosť na inžinieroch, ktorí ho tvoria – na softvérových inžinieroch. Potreba explicitne vyjadriť odporúčania správneho konania viedla odborníkov z IEEE Computer Society a ACM, aby sa pokúsili spísať etický kódex profesie softvérového inžinierstva. Kódex má 8 bodov - princípov. Prvý z nich sa týka vzťahu softvérového inžiniera k verejnosti. Táto esej sa bude zamýšľať práve nad týmto vzťahom, nad zodpovednosťou softvérového inžiniera a nad tým, ako by mal správne – eticky – konať.

Prv, než môžeme začať premýšľať o etike a profesionálnej zodpovednosti softvérového inžiniera, treba si ujasniť, čo je to etika. Podľa [3] je etika „sústava noriem o mravnom, morálnom a spoločensky vhodnom správaní ľudí“. Čo je ale morálne, etické správanie? Táto otázka je stará ako ľudstvo samo a filozofovia a teológovia sa na ňu snažili odnepamäti nájsť odpoveď. Napriek tomu ani v súčasnosti sa nedá jednoznačne zodpovedať. Preto by bolo vhodné ujasniť si základné uhly pohľadu na etiku. Etické chovanie sa dá definovať troma spôsobmi [2]:

- Etický človek je taký, ktorý má istú množinu hodnôt a žije podľa nich
- Etický človek je taký, ktorý má istú množinu hodnôt, podľa ktorej žije a ktorú zdieľa so skupinou ľudí

- Etický človek je taký človek, ktorý žije podľa množiny hodnôt, ktoré sú univerzálne platné.

V každom prípade je etika množina hodnôt, podľa ktorých by etický človek mal žiť.

Prvá definícia je najvoľnejšia. Napriek tomu, že táto definícia má mnoho prívržencov, jej interpretácia je príliš široká. Každý človek žije a koná podľa nejakých hodnôt. Ak tieto hodnoty nie sú navzájom v logickom spore alebo si vnútorne neodporujú, tak spĺňajú prvú definíciu etiky. Podľa toho každý človek, čo nie je bláznom či psychopatom, je etický človek – takže aj zlodej či profesionálny vrah. Je prirodzene citeľné, že takáto definícia etiky je nevyhovujúca. Keď nie je možné, aby sa dalo pomocou etiky vymedziť správne a nesprávne správanie, aby bolo možné mať etiku ako nejakú základňu na posúdenie správania a následného (seba)ohodnotenia, tak etika je v princípe zbytočná a zaoberanie sa ňou len márnením času. Ak je každé logické správanie správne, potom de facto neexistuje nesprávne správanie, takže zaniká potreba rozmýšľať nad tým, aké správanie je správne. Uvedomiť si toto je dôležité najmä v poslednej dobe, kde relativizácia hodnôt je na dennom programe.

Druhá definícia je jasnejšia a už umožňuje posúdenie správania sa ľudí, umožňuje rozhodnúť, aké správanie je správne alebo nie. Etiku definuje vo vzťahu človeka k ostatným ľuďom, k spoločnosti, v ktorej žije. Etický človek dodržuje pravidlá dohodnuté a vlastné spoločnosti, náboženstvu či profesii, ktoré spoluvytvára. Takáto definícia by sa už mohla zdať postačujúca. Problém je, že pravidlá sú určené spoločnosťou. Takže, v spoločnosti, kde by bolo správne podplácať, bolo by to aj etické. Respektíve, v nejakej diktatúre, kde by štát vykonával tvrdú perzekúciu voči vlastným občanom, by vojak či policajt, ktorý by sa zúčastňoval na takomto konaní, ale konal by presne podľa daných predpisov svojej profesie a podľa miestnych zákonov, konal eticky. Zase je zrejmé že akosi takéto poňatie etiky už nezodpovedá našej prirodzenej predstave o etickom správaní.

Tretia definícia hovorí, že etické princípy sú univerzálne, transcendentné a je jedno, v akej situácii či spoločnosti je človek, budú tam platiť naďalej. Takéto chápanie etiky nám umožňuje vyjadriť sa a zaujať postoj aj ku konaniu v inej krajine či profesii. Problémom môže ale nastať takzvané „miešanie sa do vnútorných záležitostí“ a problém narušenia súkromia a konflikty, ktoré z toho môžu vzniknúť. Cieľom tejto eseje nie je zachádzať do politiky, ale zamyslieť sa nad profesionálnou etikou softvérového inžiniera. Ten sa môže pri realizácii svojho povolania dostať do etického sporu. Môže nastať situácia, keď zistí, že to, čo koná, na čom sa podieľa, či to, o čom vie, je v rozpore s takto chápanou etikou, s jeho svedomím.

Etika inžiniera

Softvérový inžinier je zodpovedný svojmu zamestnávateľovi aj svojmu klientovi. Čo však, ak sa dozvie, že projekt, na ktorom robí, je, či už úmyselne alebo nie, proti spoločenskému dobru, proti životnému prostrediu, môže ohroziť život, či zdravie iných. Podľa mňa, ak softvérový inžinier uznáva univerzálnosť etických princípov, by nie mal, ale musí začať konať, niečo proti tomu podniknúť. Samozrejme, mal by dodržať hierarchiu a systém zodpovednosti, v ktorom sa nachádza. To znamená, že ak sa dozvie, že napríklad testy softvéru sú nedostatočné, mal by sa obrátiť na svojho priameho šéfa. Najprv by sa mal snažiť vyriešiť problém v rámci organizácie. Len v prípade, že je tento prístup nepostačujúci, keď nie je vôľa zo strany vedenia organizácie problém riešiť, resp. keď má opodstatnené vedomie, že takáto vôľa by nebola, musí sa rozhodnúť, čo ďalej.

Výstižný príklad držania sa etických princípov inžinierom a to aj napriek vedeniu firmy je dobre známy prípad explózie raketoplánu Challenger [4], ktorý bol dôsledkom nesprávnych manažérskych rozhodnutí a ignorovania problémov vedením oznámených inžiniermi, ktoré viedli k explózii. Autor článku, Roger M. Boisjoly, nie je softvérový inžinier, ale téma vzťahu inžiniera k verejnosti a k verejnému dobru je omnoho širšia ako oblasť softvérového inžinierstva a etický prístup musí platiť všade. Autor bol senior inžinierom vo firme MTI, ktorá stavala pre NASA raketoplány. Autor uvádza, že od začiatku boli problémy so znovupoužiteľnými prídavnými raketovými motormi, ktoré sa stále zväčšovali. Hlavný problém bol v kĺbových spojoch. Tieto, ak bol materiál pred štartom vystavený teplote nižšej ako 12 stupňov Celzia, sa nesprávali funkčne a ničili sa. Potenciálne hrozilo pri nízkych teplotách ich totálne narušenie ústiace do explózie. Autor problém nahlásil vedeniu, ale to problém ignorovalo. Aby sa problém mohol začať vôbec riešiť a bola vytvorená pracovná skupina zaoberajúca sa problémom, musel autor vyvinúť veľmi intenzívnu snahu. Skupina však mala minimálnu podporu od vedenia a riešenie problému sa predlžovalo. Napriek tomu bolo medzičasom vykonaných niekoľko, našťastie úspešných štartov raketoplánov. Osudný štart Challengeru sa konal v Januári 1986. Na deň štartu bola predpoveď počasia –8 °C, čo bolo výrazne pod požadovanými 12 °C. Na porade sa snažil autor s kolegami vysvetliť vedeniu, že v takomto prípade je štart nepredstaviteľný a musí byť odložený. Vedenie chcelo, aby sa štart za každých okolností konal. Vedenie totiž bolo pod tlakom NASA, ktorá požadovala od dodávateľov, aby sa dokazovala nie funkčnosť projektu, ale neprítomnosť nefunkčnosti. Správa o probléme, ktorý by mohol oddialiť štart, by pokazila vzťahy s NASA. Vedenie sa preto rozhodlo pre inú interpretáciu faktov, odmietlo inžinierske vysvetlenie a dalo správu pre NASA, ktorá tvrdila, že sú nejaké minoritné problémy, ale ináč je všetko OK a odporučila štart. Autor samozrejme nesúhlasil a bol z toho znechutený. Nasledujúci deň nastala smutne známa udalosť.

Po začatí vyšetrovania prípadu prezidentskou komisiou vedenie odkázalo všetkým zamestnancom, aby odpovedali veľmi stručne a v zhode s tvrdeniami vedenia. To sa snažilo ponúknuť iné dôvody explózie, aby chyba nepadla na nich. Autor napriek tomu povedal výboru celú pravdu, ako sa aj v minulosti sťažoval vedeniu a podložil to aj dôkazmi, čím si vyslúžil nevôľu a nepriazeň od vedenia. Nasledovalo ohováranie, osočovanie a jeho izolácia v rámci firmy. Vznikla voči nemu nepriateľská atmosféra a aj kolegovia ho obviňovali, že ničí záujmy firmy. Napriek tomu firma sa tvárila pred výborom, že je stále v strede diania a že s ním rátajú. Všetky rozhodnutia sa však diali mimo neho a atmosféra už bola taká zlá, že sa to nedalo vydržať a musel nakoniec dať výpoveď.

Na príklade Rogera M. Boisjolyho vidno, ako tento inžinier sa snažil chovať eticky, podľa vyšších, univerzálnych etických princípov, čím sa však dostal do sporu so svojím zamestnávateľom, voči ktorému by mal zachovávať lojalitu. Uprednostnil však univerzálny, vyšší etický princíp (verejné blaho) pred nižším etickým princípom (lojalita k zamestnávateľovi), čo malo pre neho nasledujúce nevýhody – strata dobre plateného zamestnania, koniec v kariére, osočovanie a ohováranie v médiách.

Dá sa ukázať, ako Roger M. Boisjoly intuitívne dodržal viaceré z vyššie uvedených pravidiel vzťahu inžiniera k verejnosti:

- Odmietol schváliť štart raketoplánu vediac, že sa môže skončiť katastrofou (1.03)
- Keď prišla vyšetrovacia komisia, odhalil celé pozadie problémov (1.04)
- Nebol ochotný zavádzať a uvádzať nepravdivé informácie pred vyšetrovacou komisiou (1.06)
- A napísal esej o svojom príbehu, čím výrazne prispel do diskusie o etike inžiniera (1.05)

Aj keď tento prípad sa netýka softvérového inžinierstva ako takého, týka sa inžinierstva vo všeobecnosti. Dá sa však veľmi elegantne interpolovať aj na softvérové inžinierstvo. Každý softvérový inžinier sa môže ocitnúť v situácii, keď sa dostane do rozporu medzi svojim svedomím a zamestnávateľom. V takom prípade je veľmi ťažké sa správne rozhodnúť. Správne rozhodnutie požaduje vysokú mieru osobnej integrity a často aj hrdinstva. Malo by však platiť, že uprednostnené budú univerzálnejšie princípy. V každom prípade by si mal inžinier takéto rozhodovanie dobre premyslieť a zvážiť. Niekedy totiž môže byť obtiažne zistiť, aké rozhodnutie by bolo správnejšie a vzišlo by z neho viac dobra, resp. menej zla. Vo väčšine prípadov nejde totiž o život, ako v prípade Challengeru, a každá možná akcia so sebou nesie niečo dobré a niečo zlé, s rôznym dopadom na rôzne skupiny ľudí.

Problémom etiky je, že sa týka iných ľudí. Ak by sa človek nachádzal na pustom ostrove, kde by okrem neho nebolo živej duše, nemusel by si robiť starosti s etikou. Keďže však žijeme v spoločnosti, kde sa stretajú rôzne záujmy a rôzni ľudia,

môžeme sa dostať do situácie, kde je veľmi ťažké zistiť, ktoré z možných rozhodnutí je etické. V situácii, keď má človek, softvérový inžinier, problém s takýmto rozhodnutím, môže byť vhodné, aby sa obrátil o pomoc a prediskutoval problém s kolegami z jeho stavovskej organizácie, ktorí nie sú do prípadu priamo zaangažovaní [5]. Tí mu môžu dobre poradiť a následne dať aj podporu pri realizácii daného rozhodnutia a ochrániť ho pred prípadnou perzekúciou.

Mnohí ľudia zastávajú názor, že etický problém nie je ich problém, ale problém zamestnávateľa. Svoju povinnosť k morálke si vyriešia maximálne nahlásením problému šéfovi, často ani to nie, a potom sa ich už problém netýka. Zodpovednosť má šéf, už to nie je môj problém, ja mám ruky čisté, a keď niečo, „do pekla“ pôjde šéf, nie ja. Podľa mňa je takýto prístup obyčajným pokrytectvom. Inžinier je človek, ktorý vytvára hodnoty. Produkt jeho činnosti budú využívať iní ľudia a to možno dosť dlho. Inžinier je priamo zodpovedný za výsledok, ktorý vytvorí, samozrejme aj spolu s organizáciou, pod ktorou produkt vytvára. V každom prípade je jeho zodpovednosť istá. Ako člen organizácie, ktorá produkt vytvára, je inžinier taktiež spoluzodpovedný za celý produkt, hoci jeho zodpovednosť je menšia ako manažmentu, ktorý robí strategické rozhodnutia. Preto výhovorka, že šéf o tom vie, môj problém to nie je, je zavádzajúca. Ak by som ja vedel, že pracujem na projekte, ktorý bude zneužitý na niečo, čo je v rozpore s verejným dobrom, zákonmi, alebo nebude mať dodržané normy a kritéria bezpečnosti, mal by som problém spávať s kľudným svedomím, aj ak by som šéfa na problém upozornil. Neviem vopred povedať, ako by som situáciu riešil, ale nemohol by som spokojne pracovať. Lebo inžinier nie je zodpovedný len za svoju prácu, svoj „kus kódu“, ale celkovo za produkt k celej spoločnosti. A ako človek by sa mal snažiť správať správne a eticky. Preto aj prvý bod Kódexu etiky softvérového inžiniera vraví o jeho vzťahu k verejnosti, kde sa aj píše, že softvéroví inžinieri majú *„Odhaliť príslušným osobám a autoritám každé skutočné alebo potenciálne nebezpečenstvo pre užívateľa, verejnosť alebo životné prostredie, o ktorom odôvodnene veria, že je spojené so softvérom alebo so súvisiacimi dokumentmi.“* (bod 1.04).

Záver

Zo súčasnej spoločnosti sa vytráca ideológia, hodnoty sa relativizujú. Rovnako aj pojem etika sa vykladá rôzne. Napriek snahám o jeho zrelativizovanie ostane, dúfam, chápaný ako množina *univerzálnych* hodnôt, ktoré vravia, ako žiť a konať správne. Pre softvérového inžiniera z toho vyplýva dôsledok, že zodpovedný nie je iba svojmu zamestnávateľovi, ale aj verejnosti, spoločnosti ako celku. Tieto dve zodpovednosti inžiniera môžu doviesť do konfliktu svedomia, ktorý sa môže zdať ťažko rozriešiteľný a prípadné riešenie ako bolestivé. Morálny inžinier by si mal správne rozriešenie sporu dobre premyslieť a následne sa nebáť konať v súlade s morálkou, verejným záujmom. Pomôcť by mu v tom mohli stavovské organizácie.

Použitá literatúra

[1] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[2] A. S. Gunn, P.A.Vesilind: "Enviromental Ethics For Engineers". Lewis Publishers, USA, Michigan 1988.

[3] Mária Ivanová-Šaligová: Slovník cudzích slov pre školu a prax. Slovenské pedagogické nakladateľstvo, Bratislava 1988.

[4] Roger M. Boisjoly: Ethical Decisions - Morton Thiokol and the Space Shuttle Challenger Disaster. (URL http://onlineethics.org/essays/shuttle/bois.html)

[5] Stephen H. Unger: Help for those, who need it. IEEE Ethics Committee 1995, (URL http://www.institute.ieee.org/INST/dec95/ethics.html)

Hneď ako dočítal esej, vyhľadal si v sieti informácie o spomínanej katastrofe raketoplánu Challenger. Esej ho celkom zaujala a musel pripustiť, že tvorcovia etického kódexu jasne identifikovali ďalekosiahly dosah softvéru na spoločnosť. Veď aj dnes, keď sú dopravné diaľnice, aeroautá či teleportové stanice plne riadené technikou, platí princíp zodpovednosti voči spoločnosti rovnako, ako platil na začiatku tisícročia.

Ešte raz si prelistoval prvú esej a do poznámkovej pamäte si zaznačil jej stručnú charakteristiku. Vtom si spomenul na jedenej prípad tak asi spred desiatich rokov. Istá obchodníčka z Filipín si po dlhodobom používaní svojho osobného komunikátora začala myslieť, že je potomkom vráskavca obrovského a mala neprekonateľné nutkanie simulovať Brownov pohyb. Výbor na ochranu softvérových používateľov jasne preukázal, že vážnu psychickú poruchu u nej vyvolal nový hlasový modul jej používateľskej príručky, ktorý nebol náležite otestovaný. Podobné viac či menej kuriózne prípady sa začali v tom čase objavovať nápadne často, a preto Medzinárodná softvérová asociácia (ISA) iniciovala schválenie zákona o testovaní softvérových produktov, ktorý prinútil softvérové firmy testovať svoje produkty podobne ako to robili dovtedy len farmaceutické firmy. Wesly si jasne spomínal na celosvetové odmietavé reakcie hlavne zo strany softvérových gigantov, ale verejnosť bola značne pobúrená, a preto museli ustúpiť. Asi najväčšou akciou ISA v súvislosti so softvérom a jeho vzťahom na spoločnosť bolo zavedenie prísnej kontroly nad používaním virtuálnej reality. Virtuálna realita bola ilegalizovaná a prehlásená za potenciálnu drogu, lebo prieskumy ukázali, že ľudia pri používaní virtuálnej reality strácajú pojem o skutočnosti a nemajú vôľu žiť v skutočnosti. Na druhej strane rozvoj mozgových softvérových technológií výrazne zlepšil výkonnosť ľudí pri plnení náročných úloh v stresových situáciách. Wesly mal rád hlavne tie relaxačné prístroje, ktoré pomocou ovplyvňovania mozgových vĺn umožňovali človeku hlbokú relaxáciu a meditáciu. Keď sa ešte venoval programovaniu, v zamestnaní často používali aj tzv. Super-learning prístroje, s pomocou ktorých sa dala extrémne rýchlo

naštudovať nová technológia. Za jeho čias boli mozgové prístroje (brain-machines) novinkou, ale dnes sú už bežnou súčasťou programátorskej komunity. Tak či onak, práve tieto prístroje a hlavne ich softvér sú veľkým rizikom pre ľudí, lebo pomocou nich sa dá významne ovplyvniť správanie sa človeka.

Dopil zvyšok írskej kávy a chcel sa pustiť do čítania ďalšej eseje, keď vtom sa mu zrazu v miestnosti objavil jeho osobný asistent, aby mu pripomenul, že má užiť predpísaný liek. Hmmm zamrmlal si popod nos , človeka nenechajú ani chvíľku na pokoji. Otvoril zásuvku na písacom stole a vybral z nej lieky. Jeho asistent vyzeral ako zmiešanina ľadového medveďa s japonským turistom, ale jemu sa takto páčil. Bol vtipný a zaujímavý (mal v sebe vstavaný čip AHP – almost human personality), ale niekedy až príliš dôsledný. Wesly zauvažoval či ho nezmení, ale vzápätí sa znovu pozrel na obraz na stene a začítal sa do ďalšej eseje.

Priateľ udavač

Michal Kucbel

Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
kucbel@decef.elf.stuba.sk

Abstrakt. Oblasť softvérového inžinierstva patrí medzi najmladšie medzi inžinierskymi disciplínami. Tak ako všetko nové, ešte sa musí vyvíjať a dozrievať. Základom konania každého človeka je etika. Na to, aby softvérové inžinierstvo nebolo domom postaveným na piesku slúži etický kódex softvérového inžinierstva.

Úvod

Etický kódex bol vytvorený organizáciami IEEE Computer Society a ACM ako jeden zo základov profesionality softvérových inžinierov. Obsahuje etické a profesionálne záväzky softvérového inžiniera a má sa používať ako štandard alebo návod pri vyučovaní a v praxi. Určuje tiež zodpovednosť inžiniera voči verejnosti.

Etický kódex softvérového inžinierstva ACM a IEEE, 2. princíp: Zákazník a zamestnávateľ [5]

Softvérový inžinier má konať v záujme svojho zamestnávateľa a svojho klienta a v súlade s verejným záujmom. Softvérový inžinier:

2.01. poskytne služby v rozsahu svojich možností, pričom čestne prizná ohraničenia vyplývajúce z nedostatku svojho vzdelania alebo skúseností,
2.02. nebude vedome používať nelegálne alebo neeticky získaný softvér,
2.03. nebude využívať majetok svojho zamestnávateľa alebo klienta bez ich vedomia alebo bez ich súhlasu,
2.04. vždy sa uistí či všetky dôležité dokumenty, s ktorými príde do styku, boli schválené, prípadne požiada niekoho, kto je tým poverený, aby dokument schválil,

2.05. zachová v tajnosti všetky dôverné informácie, ku ktorým sa dostane počas svojej práce, ale len v prípade, ak to nie je v konflikte s verejným záujmom alebo v rozpore so zákonom,

2.06. pohotovo identifikuje, zdokumentuje, zozbiera dôkazy a podá správu zamestnávateľovi alebo zákazníkovi, ak je presvedčený o možnom zlyhaní projektu, o možnom porušení práv duševného vlastníctva alebo sa mu projekt javí finančne nevýhodný či inak problematický,

2.07. identifikuje, zdokumentuje a podá správu zamestnávateľovi alebo zákazníkovi o dôležitých sociálnych otázkach, ktoré si všimol a ktoré súvisia so softvérom a k nemu prislúchajúcej dokumentácii,

2.08. neprijme žiadny vedľajší záväzok, ktorý by narúšal jeho výkon v jeho hlavnom zamestnaní,

2.09. nepodporí žiadne záujmy, ktoré by mohli škodiť jeho zamestnávateľovi alebo zákazníkovi, pokiaľ to nie je v záujme vyššieho etického záujmu; v takom prípade musí o tomto vyššom etickom záujme informovať svojho zamestnávateľa alebo svoje vedenie.

Čestnosť

Viaceré body v tejto časti etického kódexu by sa dali stručne vyjadriť aj jedným slovom a tým je čestnosť. Potreba mať túto vlastnosť (byť čestný) nie je vôbec špecifická pre inžinierske disciplíny, ale je vlastná všetkým ľuďom. Otázka, čo za čestné môžeme považovať a čo nie, tu bola dávno pred tým, ako prišla priemyselná revolúcia a s ňou rozvoj všetkých inžinierskych disciplín. V súčasnosti sa pojem čestnosti chápe dosť jednoznačne a toto chápanie môže byť prenesené aj do oblasti softvérového inžinierstva.

Klamanie, kradnutie alebo nedodržanie sľubov asi nikto nepovažuje za čestné. Ale keď hovoríme o klamstve, treba povedať, že aj neúplná pravda je klamstvo. Keď hovoríme o krádeži treba povedať, že všetko čo má finančnú hodnotu, je tovar, a preto aj kopírovanie softvéru je krádež. Rovnako aj narábanie s cudzím majetkom bez vedomia majiteľa je vlastne krádež. A nakoniec, keď hovoríme o nedodržaní sľubov, musíme si uvedomiť, že aj ten, čo nedokáže plniť svoj záväzok voči zamestnávateľovi, vlastne porušuje niečo ako sľub. Preto, ak je niekto čestným človekom, je predpoklad, že bude aj čestným softvérovým inžinierom.

Časti etického kódexu, ktoré ma zaujali väčšmi ako ostatné, sú časti o narábaní s informáciami v inžinierskych projektoch. Patrí k tomu utajovanie dôverných informácií, získavanie informácií o možnom zlyhaní projektu alebo informácií o konflikte medzi projektom a verejným záujmom. Nepriamo sa týchto častí týka aj záväzok nepodporovať záujmy poškodzujúce zamestnávateľa, pretože niektoré informácie môžu pre zamestnávateľa byť oveľa nebezpečnejšie ako finančná strata. V týchto častiach je dosť veľký priestor na diskusiu, čo súvisí hlavne s rôznym

chápaním a ohraničením pojmov ako napríklad, čo sú to dôverné informácie alebo verejný záujem.

Prístup k informáciám

Zachovávanie firemných tajomstiev je samozrejme žiaduce v bežných prípadoch, pretože to pomáha zachovávať autorské práva a "know-how" danej firmy. Pod takýmito bežnými prípadmi rozumiem všetky tie softvérové projekty, pri ktorých síce môžu mať zamestnanci prístup k súkromným informáciám zákazníka, ktorý si produkt objednal, ale únik takýchto informácií alebo iných informácií súvisiacich s týmto projektom, nemôže spôsobiť žiadne veľké škody spoločnosti ako celku. Spoločnosť v tomto prípade chápem ako skupinu ľudí (nie firmu), ktorá bude produkt vytvorený danou firmou skutočne používať, t.j. nielen zákazník, pre ktorého bol produkt vytváraný, ale aj ľudia, ktorí môžu nepriamo využívať tento produkt alebo byť od neho nejakým spôsobom závislí. V každom prípade, samozrejme, môže únik informácií viesť k finančným stratám alebo poškodeniu dobrého mena firmy, a preto sa každá komerčne zameraná firma snaží všetky citlivé informácie uchovávať v tajnosti.

V prípadoch, kedy hrozí v dôsledku zlyhania projektu vznik veľkých finančných škôd alebo veľké (nezanedbateľné) nebezpečenstvo ohrozenia fyzického alebo duševného zdravia jednotlivca alebo skupiny ľudí, potom práve informačná blokáda tvorí pre spoločnosť veľké riziko. A na to by som chcel predovšetkým upozorniť, že prehnané uchovávanie všetkých dôležitých informácií mimo dohľadu verejnosti je síce pochopiteľné z pohľadu komerčných firiem, ale pri vytváraní produktov s dôrazom na bezpečnosť je skôr potrebné zostaviť tím softvérových inžinierov z ľudí, ktorí nie sú až natoľko lojálni svojmu priamemu nadriadenému, aby nedokázali upozorniť na vážne bezpečnostné nedostatky v projekte. Týmto nechcem povedať, že zamestnanec nemá dbať na nariadenia svojho vedenia. Chcem tým povedať, že pokiaľ si všímame etiku zo strany zamestnanca, nemôže ani etika zamestnávateľa zostať bokom. Je zrejmé, že zamestnávateľ alebo nadriadený môže v snahe zvýšiť si svoj zisk alebo iný svoj prospech dotlačiť aj svojho zamestnanca do situácií, kde morálne správne riešenie je to najnevýhodnejšie. Etický kódex zamestnávateľa by mal preto obsahovať nasledujúce pravidlo - zamestnávať ľudí, ktorí dokážu vyjadriť svoj kritický názor v každej situácii.

Popri existencii etického kódexu softvérového inžiniera existuje ešte etický kódex človeka, ktorý je prvoradý v každej situácii. V prípade, že uznávame nadradenosť ľudského etického kódexu nad ostatnými (napríklad aj profesionálnym kódexom), môže vzniknúť konfliktná situácia, kedy budeme musieť niektoré pravidlo porušiť. Možno je slovo "musieť" trochu silné a niekto by mohol namietať, že by bolo vhodnejšie použiť slovné spojenie ako napríklad "konfliktná situácia, ktorá zvádza k porušeniu profesionálnych etických pravidiel". Situácia, ktorá je

morálne ťažko zvládnuteľná, väčšinou vedie ľudí k riešeniu, ktoré je pre ich spolupracovníkov a firmu prijateľnejšie, pričom sa prehrešujú proti pravidlám ľudskosti. Nie je to nič zvláštne, najmä ak nie je na trhu práce dostatok pracovných príležitostí na to, aby človek mohol slobodne zmeniť svoje zamestnanie. Ľudia konajúci v súlade so záujmom verejnosti sa totiž často dostávajú do problémov vo vlastnej firme a vo svojom najbližšom okolí. Príkladom môže byť aj verejnosti dobre známy prípad výbuchu raketoplánu „Challanger“ z januára roku 1986 [1]. V tomto prípade vznikli pochybnosti o bezpečnosti pripravovaného štartu raketoplánu, ale varovania technikov sa nebrali veľmi vážne. Po celej afére sa ľudia nesúci za incident zodpovednosť snažili zatajiť podrobnosti prípadu. Napriek tomu sa našiel jeden technik, ktorý oboznámil vyšetrujúcu komisiu so všetkými skutočnosťami, ktoré viedli ku katastrofe. Aj keď svoje podozrenie a obavy z možného zlyhania projektu vyjadril už pri rozhodovaní o možnom štarte, bol neskôr po nehode pre svoj nemenný postoj ku problému považovaný za udavača a musel čeliť tlaku najvyššieho predstavenstva firmy, v ktorej pracoval. Veľa ľudí môže považovať zverejňovanie takýchto informácií po incidente za zbytočné, ale nie je to celkom tak. Je pravdou, že keby boli pochybnosti technikov zverejnené ešte pred štartom, nemuselo by k nehode vôbec dôjsť. Na druhej strane informácie o tom, prečo projekt zlyhal technicky a morálne viedli k zmenám vo vesmírnych projektoch, ktoré potom nasledovali.

Ako už bolo spomenuté, vynášanie všetkých informácií na verejnosť nie je pre firmy žiaduce, preto by sa mali snažiť vytvoriť si vlastný kontrolný orgán, ktorý by nezávisle posudzoval firemné projekty [4]. Nie je asi možné niečo také docieliť vo firme, ktorá je čo do počtu zamestnancov príliš malá a celá jej hierarchia nemá viac ako dve až tri úrovne, ale vo väčších firmách by mohli existovať autority, ktoré nie sú priamo zainteresované do projektu, a ktoré by mohli podporiť ľudí pracujúcich na projekte, o ktorom vedia, že je problematický a nenašli pochopenie pre kritiku u svojich priamych nadriadených.

V určitých projektoch by som dokonca nepovažoval žiadnu komerčnú firmu za dôveryhodnú. Napríklad projekty v oblasti energetiky, hromadnej dopravy alebo vojenské projekty si vyžadujú dohľad autority, ktorá nie je finančne ani morálne angažovaná na projekte, a ktorá je neustále pod dohľadom spoločnosti. Takou autoritou by mal byť štát, pokiaľ je takýto dohľad v súlade so zákonom alebo by to mala byť iniciatíva obyčajných civilných ľudí v prípade, keď je dohľad zo strany štátu diskutabilný alebo je to v určitom zmysle priestupok voči pravidlám spoločnosti. Takýmto priestupkom samozrejme nemyslím prekročenie zákona a nijako k tomu nemienim niekoho nabádať, ale myslím si, že zachovávanie spoločenských zákonov môže niekedy viesť k porušeniu etických zákonov.

Z dôvodu možného nedostatku kritických ľudí v niektorých tímoch nie som v zásade ani proti ľuďom označovaným ako hackeri a proti činnosti, ktorú robia. Musím tu ale hneď uviesť, že sa v žiadnom prípade nestotožňujem s ľuďmi, ktorí sa považujú za predurčených na to, aby niekomu poškodzovali údaje alebo prostredníctvom počítačov obťažovali ľudí. Takýchto ľudí za hackerov nepovažujem.

Ako hackera chápem človeka, ktorý získava informácie, ktoré sú iným nedostupné a ktoré môžu predstavovať spoločenské nebezpečenstvo [3]. Získavaním informácií sa ale jeho úloha nekončí, preto je dôležité aj to ako s týmito informáciami bude ďalej narábať [2]. Rozhodne by sa nemal snažiť získané informácie zneužiť, ale mal by na ne upozorniť zodpovedné osoby, ktorých sa informácie týkajú a, až keď narazí na ich nezáujem, má právo obrátiť sa na verejnosť. Je tu samozrejme namieste otázka, či je možné takúto činnosť tolerovať, keď je riziko, že sa hacker dostane aj k súkromným alebo veľmi osobným informáciám ľudí. Ale na druhej strane ak sa na problém pozeráme z nadhľadu, musí byť jasne vidieť nepomer medzi rizikom škôd spôsobených zverejnením súkromných informácií a rizikom, že spoločnosť sa nedozvie včas informácie, ktoré by právom mala vedieť. Zverejňovanie závažných informácií môže síce vyvolať šírenie poplašných správ, ale to nie je dôvod na to, aby sa verejnosti odopieralo právo na informácie. Ak predsa viem, že vlastním niečo, čo patrí niekomu inému (v tomto prípade informácie), nemôžem mu toto vlastníctvo odoprieť, ani ak si myslím, že s týmto majetkom nevie správne zaobchádzať.

Ľudia, ktorí zneužívajú známe diery v operačných systémoch na tvorbu vírusov by nemali byť označovaní za hackerov. Hacker v podstate môže nahradiť človeka, ktorý vie kriticky posúdiť nebezpečenstvo projektu, na ktorom pracuje, a ktorý by mal byť v každom tíme vyvíjajúcom bezpečnostne kritické aplikácie prítomný.

Udržiavanie citlivých informácií v tajnosti môže pri niektorých projektoch viesť ku vzniku situácie, ktorá môže byť nejakým spôsobom nebezpečná, ale na strane druhej by povinnosť zverejňovať všetky informácie narúšala práva duševného vlastníctva a poškodzovala firmy v ich konkurenčnom boji. Jazýčkom na váhach, kedy ponechať zverejňovanie informácií len na rozhodnutí firmy, ktorej sa to týka, a kedy naopak striktne vyžadovať verejný dohľad nad projektom, by mala byť miera dopadu daného projektu pre spoločnosť v prípade, že projekt zlyhá alebo bude zneužitý.

Použitá literatúra:

[1] Vivian Weil: Whistleblowing: What have We Learned Since the Challenger? Illinois Institute of Technology, August 1996.

[2] Anderson Silva: Hacker: Genius or Criminal? April 1998.

[3] Dorothy E: Concerning Hackers Who Break into Computer Systems. 13th National Computer Security Conference, Washington, Oct. 1-4, 1990.

[4] Walter L. Elden: "Carrot" and "Stick" Incentive Proposals for Eliminating Whistleblowing.
P.E., IEEE, NIEE, CPSR, August 1996.

[5] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

Wesly pozrel na hodinky a zistil, že je už veľa hodín. Rozhodol sa, že sa zo zvyškom esejí musí viac poponáhľať. Nechcel strácať čas, a tak sa pustil do čítania ďalšej. S prekvapením zistil, že sa venuje tej istej téme ako esej predošlá.

Úprimnosť, čestnosť či väzenie

Richard Koncz
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
koncz@decef.elf.stuba.sk

Abstrakt. Nasledujúca esej je zovšeobecnením mojich skúseností a názorov na etický kódex softvérových inžinierov vydaný organizáciou ACM a IEEE [1], konkrétne na princíp týkajúci sa zákazníka a zamestnávateľa. V prvej časti opisujem moje skúsenosti s jednotlivými pravidlami kódexu, čím sa snažím na konkrétnom príklade vysvetliť ich podstatu. Druhá časť sa zaoberá prípadom Jiřího Herolda, ktorého odsúdili za používanie nelegálneho softvéru.

Etický kódex softvérového inžinierstva ACM a IEEE, 2. princíp: Zákazník a zamestnávateľ [1]

Softvéroví inžinieri by mali konať spôsobom, ktorý je najvhodnejší pre zákazníka a zamestnávateľa a zároveň je v zhode so spoločenským záujmom. Zvlášť by mali dodržiavať nasledovné pravidlá:

2.01. Vykonávať prácu v rámci kompetencií, byť čestný a úprimný pri vyjadrení svojich znalostí a skúseností.
2.02. Vedome nepoužívať softvér, ktorý je zaobstaraný alebo udržiavaný nelegálne a neeticky.
2.03. Narábať s majetkom zákazníka a zamestnávateľa len spôsobom, na ktorý sú splnomocnení a len s vedomím a súhlasom zákazníka alebo zamestnávateľa.
2.04. Uistiť sa či ľubovoľný dokument, na ktorý sa spoliehajú, bol schválený a, ak sa to požaduje, dať ho schváliť príslušnej osobe na to poverenej.
2.05. Udržať v tajnosti dôverné informácie získané počas profesionálnej práce, pričom dôvernosť musí byť v súlade so spoločenským záujmom a zákonom.
2.06. Identifikovať, zdokumentovať, zozbierať údaje a dôrazne oznámiť zákazníkovi alebo zamestnávateľovi ak, podľa ich názoru, projekt pravdepodobne zlyhá, značne sa zvyšujú jeho náklady, porušuje zákon duševného vlastníctva alebo je iným spôsobom problematický.
2.07. Identifikovať, zdokumentovať a oznámiť dôležité spoločenské otázky, ktorých sú si vedomí, v softvéri a k nemu pridruženej dokumentácii zákazníkovi alebo zamestnávateľovi.

2.08. Neprijímať žiadne vedľajšie záväzky, ktoré by znížili výkon práce, ktorú vykonáva u hlavného zamestnávateľa.
2.09. Nepodporovať žiadne záujmy zamerané proti zamestnávateľovi alebo zákazníkovi, pokiaľ si to nevyžaduje vyšší etický princíp. Ale aj v takomto prípade musí informovať svojho zamestnávateľa alebo inú autoritatívnu osobu.

Škola a práca zároveň

Ako čerstvý študent inžinierskeho štúdia som si jedno ráno povedal, že prišiel ten čas, keď je nutné overiť si svoje nadobudnuté teoretické vedomosti v praxi. Po hodinách a dňoch strávených čítaním a odpovedaním na inzeráty v novinách a na Internete sa dostavili prvé ponuky. Z pestrej palety pracovných príležitostí však rýchlo vypadli, hoc aj nádejní, kandidáti vyžadujúci plný pracovný úväzok. A tak som sa dostavil na pohovor do istej firmy, ktorá svojou koncepciou a zameraním sľubovala naplnenie mojich snov o budúcom zamestnaní. Vtedy som ešte netušil, že moje sklamanie bude veľké a tobôž nie o napĺňaní nejakých kódexov etiky.

Úprimnosť nadovšetko

Na pohovor som sa dostavil naladený podať maximum a dokázať aj sám sebe, že som dozrel na inžiniera a roky strávené v škole neboli zbytočným mrhaním času. Príjemne ma prekvapilo prostredie budúceho zamestnania i spolupracovníci, ktorý mi už od prvej chvíle dali najavo, prečo som tu a čo sú moje povinnosti. Zároveň však pristupovali ku mne ako k rovnocennému partnerovi. Bol to mladý kolektív, ktorý mi mal byť zároveň zamestnávateľom i budúcimi spolupracovníkmi.

Po úvodnom pohovore mi predostreli projekt, na ktorom som mal pracovať a oboznámili ma so základnými faktami. Nemal som až také skúsenosti s technológiou, ktorá sa pri ňom využívala, ale bola to zároveň pre mňa aj výzva. Už od prvej chvíle som mal pocit, že tento projekt je veľmi veľký a časovo rozsiahly. Mojou vstupnou previerkou mala byť analýza danej problematiky, určenie zdrojov a prostriedkov potrebných k danému projektu. Po prvotnej analýze a príprave základných požiadaviek som nadobudol dojem, že tento projekt je natoľko rozsiahly, že jeho ukončenie som odhadol na minimálne mesiac neskôr, ako bolo pôvodne plánované. Na ďalšie stretnutie som prišiel s dokumentáciou, ktorá okrem analýzy a bližšej špecifikácie požiadaviek a zdrojov obsahovala aj predbežný plán prác a časový harmonogram. Na moje veľké prekvapenie mi však oznámili, že som na projekte zatiaľ sám, lebo druhá osoba je z dôvodu zaneprázdnenia z kola von.

Nastal čas urobiť rázne rozhodnutie z mojej strany. Až teraz viem, že som konal v zhode s pravidlom číslo 2.06. Oznámil som zamestnávateľovi, že v prípade ak chce dodržať aspoň mnou vypracovaný časový harmonogram prác, je nutné na tento

projekt zabezpečiť štyroch, piatich ľudí znalých v danej problematike a s prácami začať najneskôr do dvoch týždňov. Táto podmienka bola zároveň aj mojou podmienkou na uzavretie pracovného pomeru s ním. Rozišli sme sa teda s tým, že po nábore ľudí ma budú kontaktovať.

Čas plynul, ale správy od môjho nádejného zamestnávateľa neprichádzali. A tak som sa ocitol na pohovore v ďalšej firme. Na pohovor som sa dostavil s presvedčením, že budem úprimný vo veci svojich vedomostí a skúseností. Z predchádzajúcej skúsenosti som totiž usúdil, že výzva je síce pekná vec, ale úprimnosťou a priamosťou v tejto oblasti môže predísť viacerým konfliktným situáciám. Nakoniec som tú prácu dostal a nastúpil som do svojho prvého zamestnania.

Keď som sa rozhodol, že si nájdem prácu popri škole, vedel som, že do úvahy prichádza len práca na polovičný alebo čiastočný úväzok. Inak by škola išla na úkor práce alebo naopak a výsledok by bol, že ani jedno by som nevykonával naplno a dôkladne. Neuplynul ani týždeň od nástupu do práce a už som sa dostal do rozporu s ďalším pravidlom etického kódexu. Do zamestnania som totiž nastupoval tesne pred koncom semestra, keď zaťaženie mojej osoby v škole bolo veľké a najmä, schyľovalo sa k termínu odovzdania časovo náročného ročníkového projektu. Zrazu som stál pred rozhodnutím. Škola alebo práca. Vedel som, že škola je prednejšia, ale zároveň som sa obával, či týmto rozhodnutím nestratím nádejné zamestnanie. Dlho som váhal či s týmto problémom mám ísť za zamestnávateľom. Ako sa teraz s odstupom času na to pozerám, urobil som správne rozhodnutie, keď som sa rozhodol prerušiť prácu počas skúškového obdobia. Toto rozhodnutie bolo výhodné aj pre zamestnávateľa, lebo po ukončení skúšok som sa naplno vrhol do projektu, ktorého výsledok dodnes pomáha takmer každému zamestnancovi firmy pri práci.

Prípad Jiřího Herolda

„*Samosudca Obvodného súdu pre Prahu 10 vydal trestný príkaz, ktorý uznal Jiřiho Herolda, prevádzkového programátora vo firme Českomoravská lekárnická platebna, a.s. vinným z dôvodu porušenia zákona č. 35/1965 Zb. (autorský zákon) a odsúdil ho k trestu odňatia slobody na jeden rok. Prevádzkový programátor bol zodpovedný za prevádzku firemnej siete a vybavenie počítačov príslušným softvérovým vybavením s riadnymi licenciami vo firme.*“

Tento krátky výňatok z článku prílohy denníka Pravda [2] stručne naznačuje o čo v prípade Jiřího Herolda ide. Je to prvý prípad odsúdenia konkrétnej súkromnej osoby, nie firmy ako celku, za podobný prečin v celej východnej Európe. Odsúdený Jiří Herold vo svojej výpovedi uviedol: „*Bol som uznaný vinným, keďže som bol u svojho zamestnávateľa zodpovedný za prevádzku počítačovej siete, teda aj za softvér. Táto zodpovednosť nebola priamo v mojej pracovnej zmluve uvedená, ale pri vyšetrovaní celej záležitosti to uviedli moji nadriadení. Nebol som si stopercentne istý, či môj zamestnávateľ všetky programy riadne zakúpil. Rozsudok proti mojej osobe je výsledkom toho, že som tento problém nijako neriešil.*“

Dodržiavanie pravidla číslo 2.02 sa v inžinierskej obci pokladá ako samozrejmosť. Prax je však úplne iná. Podľa štatistík sa odhaduje, že až šesťdesiat percent programov je nelegálne používaných. Odhalenie podobnej trestnej činnosti je pre príslušné orgány veľmi ťažké. Postupne však prichádza k zlepšeniu tejto situácie aj u nás. Jednak uvedomovaním si jednotlivých etických princípov a zásad jednotlivými firmami ako aj úpravou legislatívy. Vznikajú organizácie dozerajúce na legalitu softvéru vo firmách i domácnostiach. U nás najznámejšou je BSA, ktorá „dozerá" nad legalitou používaní produktov firmy Microsoft. Reaguje aj na anonymné oznamy, podobne ako tomu bolo v prípade Jiřího Herolda, keď neznáma osoba podala oznámenie o nelegálnom používaní softvéru na firmu Českomoravská lekárnická platebna, a.s.. Často sa však tieto oznamy stávajú súčasťou konkurenčného boja spoločností. Tu potom vstupuje do úlohy iný etický princíp.

Precedens, ktorý vyvolalo rozhodnutie súdu v tomto prípade, určite zvýši zodpovednosť, ale i rizikovosť povolania správcu počítačovej techniky vo firmách. Dôležitým faktorom sa stane miera zodpovednosti za jednotlivé činnosti, ktoré správca vykonáva. Nastane asi výrazný posun vpred v smere určenia presných právomocí a zodpovednosti pracovníka v pracovnej zmluve, aby sa predišlo neskorším rozporom, čo bolo, a čo nie, jeho zodpovednosťou. Hovorí sa, že „*neznalosť zákona neospravedlňuje"*, ale táto odveká pravda sa asi zmení na „*neznalosť znenia pracovnej zmluvy neospravedlňuje"*. Zároveň však bude nutné vypracovať mechanizmus, ktorý umožní správcom rozhodovať a konať v prípade, že ich zamestnávateľ nekoná v súlade so zákonom, čo prinesie akési otočenie pohľadu právomocí zamestnávateľa a zamestnanca.

Záver

Táto esej je obrazom môjho ponímania jednotlivých pravidiel etického kódexu softvérového inžiniera. Veľa z týchto pravidiel je mi blízkych a vnímam ich ako samozrejmosť. Vychádzajú totiž zo základných morálnych a etických hodnôt človeka a aj v prípade, že by som tento kódex nikdy nečítal, ich dodržiavanie je pre mňa takou istou samozrejmosťou ako napríklad pozdrav na ulici. Snažil som sa poukázať najmä na tie pravidlá, s ktorými som sa vo svojej krátkej praxi stretol a ktorých naplnenie sa mi dúfam darí dodržať.

V druhej časti eseje som reagoval na konkrétnu udalosť, ktorá sa stala prednedávnom a podľa môjho názoru bude akýmsi prelomom v problematike používania legálneho softvéru na Slovensku. Zároveň poukazujem aj na druhú stránku veci a tou je postoj zamestnávateľa k zamestnancovi, ktorý v tomto prípade nebol celkom etický.

A čo dodať na záver. Len presvedčenie, že čestnosť a úprimnosť sa stanú vlastné každému softvérovému inžinierovi a nedostanú sa svojim konaním nikdy do situácie, na konci ktorej je slovko väzenie.

Použitá literatúra

[1] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[2] I. Piovarči: Odsúdený zamestnanec. Pravda – Technické spektrum, číslo 11, 2000, str. 5.

[3] Richard G. Epstein: The case of the killer robot. John Willey & Sons, 1997. (ISBN 0-471-13823-1)

[4] Jim McCarthy: Softwarové projekty. Computer Press, 1999. (ISBN 80-7226-164-0)

Weslyho zarazil v kódexe hlavne termín nelegálny softvér. Nevedel si vybaviť, čo to presne znamená, ale mal dojem, že ten termín už niekedy počul. Nevedel si spomenúť, v akej súvislosti. Po chvíľke premýšľania mu napadlo, že sa naposledy na jednom spoločenskom večierku bavil s nejakým historikom o pomeroch na začiatku tisícročia. Ten historik hovoril o tom, ako sa zvykol kradnúť softvér tým, že sa jednoducho preniesol na iné médium. Prítomní sa vtedy na večierku náramne bavili nad takou primitívnou formou kriminality. Ešte pre istotu zalistoval v archívoch, aby sa o problematike dozvedel viac. Zistil, že niekedy okolo roku 2031, keď globalizácia pokročila tak ďaleko, že softvérové zákony sa uzatvárali s celosvetovou platnosťou, sa podarilo koncernu MSLinux presadiť novelizáciu zákona o softvéri. Táto novela prinútila každého výrobcu softvéru, aby použil LTCW mechanizmus na zabezpečenie svojich produktov (samozrejme, že MSLinux vlastnil autorské práva na algoritmus). LCTW algoritmus bol objavený úplne náhodou ako vedľajší produkt pri vývoji automatickej kosačky na trávu. Bolo dokázané, že na prelomenie tohoto algoritmu neexistuje vo vesmíre dostatok energie a pritom jeho použitie bolo viac než triviálne. Tak sa vlastne skončila éra nelegálneho softvéru. Nik si už vlastne nelegálny softvér nainštalovať nemôže, lebo pri akomkoľvek pokuse ukradnúť nejaký softvér sa pošle signál do siete a jeho výrobca má hneď k dispozícii informáciu o osobe a mieste, kde sa vyskytol pokus o nelegálnu inštaláciu. Dokonca boli zavedené policajné jednotky, ktoré riešili práve takéto prípady. Postihy za takéto činy sa rôznia, ale sú tak tvrdé, že až na pár výnimiek sa o to nik nepokúša. Postupom času sa prešlo od odňatia slobody ku tvrdším trestom, ako napríklad zákaz výkonu profesie v oblasti softvéru, alebo dokonca odňatie prístupu do INTERNET-u na 1 a viac rokov.

V eseji sa znova spomínal výbuch raketoplánu. Pomyslel si, že to musela byť poriadna aféra, keď je prípad spomenutý už v druhej eseji. Tiež si všimol bod, ktorý sa venoval dôverným informáciám a ich utajovaniu. Pomyslel si, že to je problém, ktorý sa len tak ľahko nebude dať odstrániť. Veď informácie sú v dnešnej dobe asi tým najdôležitejším tovarom a je prirodzené, že existujú snahy o ich krádež alebo zneužitie. Ochrana a snaha o odcudzenie znamenali pre softvérový priemysel veľkú pohonnú silu. Každé vynájdenie ochranného systému proti úniku informácií vyvolalo odpoveď v podobe technológie schopnej nabúrať ho. Ale veľkým problémom v tejto

súvislosti sú hlavne úniky informácií spôsobené ľudským faktorom. Akýkoľvek obranný mechanizmu sa dá jednoducho obísť tak, že narušiteľ použije metódy sociálneho inžinierstva (násilné, úplatky alebo iné) a prekoná tak softvérové a iné ochranné bariéry. Preto je veľmi dôležité, aby si softvéroví inžinieri uvedomovali dôležitosť ochrany informácií. Wesly si pamätal na mnohé prípady únikov informácií, ktoré spôsobili obrovské škody. Napríklad niekedy v dvadsiatich rokoch sa stal prípad, kedy sa do rúk guatemalských extrémistických skupín dostali dôverné informácie o novej matematickej teórii chaosu, ktorá umožňuje počítačové simulácie atraktorového neuorogenézneho správania sa takmer ľubovoľného systému. Tento únik takmer spôsobil krach svetovej burzy, lebo pomocou novej teórie sa dal významne predvídať vývin akcií a cien komodít. Matne si ešte spomenul, že elektronické voxel-doláre klesli v tej dobe o neuveriteľných 0.000129 percenta, čo bol rekord poklesu od ich zavedenia. Podobných prípadov sa vyskytlo viac a je viac než pravdepodobné, že s týmto problémom sa nebudeme vedieť tak jednoducho vyrovnať ani v budúcnosti.

Wesly sa pobral do vedľajšej miestnosti a podišiel ku automatu na jedlo, ktorý nazval jeho výrobca originálne Food'o'matic. Zamrmlal niečo v tom zmysle, že je hladný a dal by si už obed. Automat Weslyho hlas prebral z celodenného ničnerobenia a po asi 3 tisícinách sekundy vypísal na obrazovku na dverách popis jeho dnešného menu. Mhhm špargľa v syrovej omáčke a cestoviny prevarené elektrónovým prúdom, to neznie zle pomyslel si. Celkom mu vyhovovalo, že v stravovaní sa o neho tento milý prístroj staral a dohliadal na vyváženosť jeho stravy. Na obrazovke klikol na tlačidlo COOK a pobral sa späť do svojej kancelárie. Na stôl si pripravil ďalšiu esej a začítal sa do nej kým sa mu pripravovalo jedlo...

Niekoľko postrehov k vlastnostiam softvérového produktu a jeho výrobe

Branislav Lehotský
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
blehotsky@pobox.sk , lehotsky@geocities.com

Abstrakt. Esejou som sa pokúsil reagovať na body tretieho princípu kódexu etiky softvérového inžinierstva organizácií ACM a IEEE. V tomto princípe sú obsiahnuté zásady týkajúce sa výrobku – produktu a niektorých aspektov jeho výroby. Pokúsil som sa rozvinúť a vysvetliť, ako som ja pochopil v ňom uvedené odporúčania.

Etický kódex softvérového inžinierstva ACM a IEEE, 3. princíp: Produkt [2]

Softvéroví inžinieri by mali zaistiť (zabezpečiť), aby ich výrobky a súvisiace modifikácie dosahovali najvyššie možné profesionálne štandardy. Podrobnejšie softvéroví inžinieri by mali primerane:

3.01. Snažiť sa o vysokú kvalitu, akceptovateľné náklady (cenu) a o vhodný plán (rozvrh), zabezpečiť aby významné základné body predmetu kontraktu („tradeoffs“) boli jasné a akceptovateľné pre zamestnávateľa aj klienta a aby brali do úvahy záujmy používateľov i verejnosti.

3.02. Zaistiť vhodné a dosiahnuteľné ciele a zámery pre každý projekt, na ktorom pracujú alebo ho navrhujú.

3.03. Identifikovať, definovať a adresovať etické, ekonomické, kultúrne, právne a environmentálne otázky súvisiace s pracovnými projektmi.

3.04. Zabezpečiť, aby boli kvalifikovaní pre každý projekt, na ktorom pracujú alebo zamýšľajú pracovať, vhodnou kombináciou vzdelania, tréningu a skúseností.

3.05. Zaistiť, aby boli použité vhodné metódy pre každý projekt, na ktorom pracujú alebo zamýšľajú pracovať.

3.06. Pracovať podľa profesionálnych štandardov (ak sú dostupné), ktoré sú najvhodnejšie pre danú úlohu. To za predpokladu, že sú eticky a technicky prijateľné.

3.07. Usilovať sa o úplné a dôkladné porozumenie špecifikácii softvéru, na ktorom pracujú.

3.08. Zaistiť, aby bola špecifikácia softvéru, na ktorom pracujú, dobre zdokumentovaná, aby spĺňala požiadavky používateľov a bola vhodne schválená.

3.09. Zaistiť realistické kvantitatívne odhady nákladov, plánovania, pracovníkov, kvality a výsledkov na každý projekt, na ktorom pracujú alebo zmýšľajú pracovať a uviesť mieru neistoty (toleranciu) spojenú s týmito odhadmi.

3.10. Zaistiť zodpovedajúce testovanie, odladenie softvéru a súvisiacich dokumentov, na ktorých pracujú.

3.11. Zaistiť zodpovedajúcu dokumentáciu obsahujúcu objavené kľúčové problémy a prijaté riešenia pre každý projekt, na ktorom pracujú.

3.12. Pracovať na vývoji softvéru a na súvisiacich dokumentoch, ktoré rešpektujú súkromie tých ľudí, ktorí prídu so softvérom do kontaktu.

3.13. Dbať na používanie len presných údajov, získaných etickými a legálnymi prostriedkami a používať ich len spôsobmi korektne autorizovanými.

3.14. Udržiavať (dbať na) integritu dát, vyvarovať sa neaktuálnych údajov a výskytov chýb.

3.15. Vykonávať všetky formy údržby softvéru s rovnakým profesionalizmom ako nový vývoj.

Softvérový výrobok a proces jeho vývoja sú často skloňované pojmy v softvérovom inžinierstve. Je teda prirodzené, že použité rady, odporučenia či pravidlá týkajúce sa podstaty výrobku a jeho „výroby“ obsahuje i kódex etiky softvérového inžinierstva. V tejto práci som skomentoval a rozviedol podľa môjho názoru hlavné črty tretieho princípu „Produkt“.

Veľa muziky za málo peňazí a jasné pravidlá (3.01, 3,02)

Aký by mal byť ideálny produkt, ktorý tím programátorov pod vedením softvérového inžiniera vytvorí? Kľúčové faktory sú **kvalita** (mal by spĺňať, resp. i prekračovať všetky požiadavky zákazníka), **čas** (vývoj by mal byť rýchly) a **cena** (mal by byť lacný). Mať, či vytvoriť dokonalý softvér vyvinutý za okamih času a zadarmo je nesplniteľný sen, o ktorom „snívajú“ obidve strany: softvérová firma i zákazník. Líšia sa prirodzene len v predstave o predajnej cene: Výrobca by ho síce rád zadarmo vyrobil, ale predal za „kráľovskú“ cenu maximalizujúcu zisk. Sem by sme ťažko „vmontovali“ etiku. Cenu však našťastie určí trh (ak ide o generický softvér), alebo tender ak ide o softvér vyvíjaný na zákazku. Rozumný „etický“ zisk by sme totiž definovali len ťažko. Čas potrebný na „výrobu“ prirodzene súvisí s rozsahom problému, ale i so schopnosťami a skúsenosťami celého vývojového tímu. Ideálne je, ak sa správne odhadne a dodrží.

Predpokladaná kvalita bude zaručená, ak zákazník i výrobca dostatočne jasne vedia, čo chcú. Pri uzatváraní zmluvy alebo kontraktu prirodzene nemôžu byť zrejmé všetky podrobnosti o produkte (tie upresní analýza), mali by však byť veľmi dobre deklarované a ujasnené základné kľúčové body. Musia byť jasné mantinely a ciele celého projektu, ktoré už svojvoľne žiadna strana počas vývoja nezmení.

Špecifikácia – špeciálne na „špici“ (3.07, 3.08)

Vytvorenie dôkladnej, dostatočne presnej špecifikácie je iste hlavným kľúčom ku kvalitnému produktu. Odzrkadľuje porozumenie problematiky a podchycuje požiadavky. Je to základný kameň celej budúcej stavby a pragmatičnosť i etika by mali softvérového inžiniera (ako analytika, resp. manažéra) viesť, aby bol položený pevne.

F. P. Brooks vo svojej knihe „Mýtický človeko-mesiac“ [1] používa pojem „manuál“ práve vo význame dôkladnej písomnej špecifikácie (teda nie ako ekvivalent používateľskej príručky). Takto výstižne ďalej charakterizuje jeho úlohu a význam:

„Manuál, alebo písaná špecifikácia je veľmi dôležitý nástroj, i keď nie sám dostačujúci. Manuál je externá špecifikácia výrobku. Opisuje a predpisuje každý detail toho, čo všetko má byť pre užívateľa viditeľné (opis všetkých rozhraní). Nemal by však opisovať to, čo používateľ nevidí! To musí zostať vecou implementátora a tu musí zostať jeho sloboda návrhu nedotknuteľnou. Architekt-návrhár by mal byť vždy pripravený načrtnúť možnú implementáciu pre nejaký prvok (element), ktorý navrhol, ale nesmie sa ju pokúšať programátorovi prikazovať a nariaďovať.

Štýl musí byť precízny, úplný a vhodne detailný. Používatelia často požadujú jednoduché definície, všetky musia preto obsahovať a opakovať všetko podstatné a všetko musí súhlasiť, 'sedieť'. To robí manuály síce zle čitateľnými a nudnými, ale precíznosť a dôkladnosť je pre ne dôležitejšia než 'zážívnosť'... “

„ ...Žiadny prirodzený ľudský jazyk nie je dôkladným a dostačujúcim nástrojom na podobné špecifikácie a definície. Preto sa musí pisateľ manuálu (písomnej špecifikácie) veľmi namáhať, keď používa prirodzený jazyk a chce dosiahnuť potrebnú dôkladnosť. Atraktívnou alternatívou je použiť pre také definície formálny zápis. Výsadou formálnych zápisov je práve presnosť.“

Formalizácia je zrejme faktor, ktorý by nám tu nemal byť ľahostajný. Špecifikácia založená na dobrej formalizácii je prehľadnejšia, exaktnejšia. Možno práve pre nedostatok vhodnej formalizácie je softvérové inžinierstvo obtiažnejšie než inžinierstvo iných disciplín. Keď máme dobre formálne podchytenú a pochopenú problémovú oblasť (doménu), s ktorou súvisí vyvíjaný softvér, bude dobre formálne špecifikovaný aj on sám.

Požiadavky pre veľké a komplexné systémy sú často ťažko formulovateľné hneď na začiatku, vyvíjajú a dopĺňajú sa počas životného cyklu produktu. Nebojme

sa teda evolúcie, úprav a zmien „už za behu“. Neetické by mohlo byť práve i jednoduchšie „skostnatelé“ držanie sa počiatočnej špecifikácie za každú cenu a ignorovanie flexibilných evolučných softvérových procesov s vedomím, že utrpí kvalita.

Špecifikácia by mala obsahovať aj základný návrh spôsobu ovládania programu (výrobku), spôsob komunikácie užívateľa s ním. Teda určenie vonkajšieho dizajnu, používateľského rozhrania.

Dobré a vhodné zdokumentovanie špecifikácie je okrem iného potrebné aj pre záverečnú kontrolu, validáciu a schválenie celej špecifikácie. Musíme sa spätne presvedčiť, či skutočne uspokojuje všetky požiadavky budúcich používateľov (zákazníkov).

Čo je mimo, to je „mimo“? (3.03)

Heuristikou úspechu je síce predovšetkým sledovanie záujmov zákazníka a našej softvérovej firmy, nie však „cez mŕtvoly“. Musíme pri sledovaní spoločných cieľov zohľadniť i okolitý svet a spoločnosť, aby pri neskoršom používaní výrobku (resp. už pri jeho vývoji) neutrpela ujmu tretia strana alebo životné prostredie. Plánované chovanie a funkčnosť programu treba teda v takých prípadoch (ak existuje konkrétny zákazník - po diskusii s ním) vhodne upraviť či obmedziť.

Dokumenty – len nut(d)ná byrokracia? ; Plánovanie (3.11, 3.09)

Z teórie vieme, že samostatné dokumenty by mala produkovať každá fáza softvérového vývojového procesu. (špecifikácia, analýza, návrh,…) Tento fakt je podmienkou dobrej „viditeľnosti“ samotného procesu. Odhaľme ďalšie aspekty, ktoré predstavuje a prináša dokumentácia. Upresnime si však najprv význam samotného pojmu. Dokumenty a dokumentáciu budeme chápať vo význame akýchkoľvek materiálov (v papierovej alebo elektronickej podobe) potrebných na „réžiu“, riadenie a viditeľnosť softvérového procesu. Môžu to byť materiály, ktoré zákazník nikdy neuvidí ale i tie, ktoré sú určené na komunikáciu s ním (Teda nechápme tu pojem dokumentácia len vo význame sprievodných materiálov dodaných zákazníkovi s finálnym produktom.)

Zjednodušene môžeme vyjadriť zmysel manažérskej úlohy ako jasné určenie a zodpovedanie otázok „čo, kedy, koľko a kto“. To platí i pri tvorbe softvéru pre softvérového inžiniera či manažéra. Musia byť stanovené ciele a podrobne špecifikovaný výrobok („čo“), musí byť vytvorený plán („kedy“), stanovený rozpočet („koľko“) a organizačná personálna schéma („kto“). Na viditeľné „podchytenie“ odpovedí na tieto otázky treba vytvárať dokumenty.

Takýto pohľad na dokumenty prirodzene úzko súvisí s témou plánovania projektu i s jeho riadením. Plán je komunikatívny a dôkladný len ak je vhodne „zachytený“ dokumentmi. Ideálny plán by mal obsahovať dokumenty, ktoré dostatočne presne odpovedajú na všetky vyššie spomínané otázky. Nutnou - iste nie len etickou úlohou manažéra je vytvoriť dobrý plán s realistickými odhadmi a zrealizovať ho. Pri odhadoch treba používať i kvantitatívne metódy, nerobiť len odhady urobené na základe tušení a skúseností.

F. P. Brooks zdôvodňuje potrebu vytvárania dokumentácie okrem iného takto (voľný preklad):

1.) Zachytenie rozhodnutí a predstáv je základné. Len keď niekto niečo zaznačí (zapíše), môžu sa objaviť medzery a „vystúpiť“ nekonzistentnosti. Samotný „akt“zapisovania (resp. vytvárania schém či grafov) si vynucuje súčasné spravenie stoviek rôznych neviditeľných „mini-rozhodnutí“. A práve ich existencia oddeľuje jasnú exaktnú „politiku“ (zámery, heuristiku postupu) od nejasných a neurčitých.

2.) Dokumenty tiež sprostredkovávajú manažérove rozhodnutia druhým, sú určitou formou komunikácie. Často tak manažér s úžasom zistí, že pre niektorých členov jeho tímu nie sú známe ani základné princípy jeho „politiky“ a zámerov. I keď jeho hlavnou starosťou je sledovanie, aby každý šiel tým istým smerom, jeho skutočnou „šéfovskou“ dennou úlohou je komunikácia, nie len robenie rozhodnutí. Jeho dokumentácia „prináša svetlo“ práve do tejto úlohy.

3.) Dokumenty dajú spätne manažérovi potrebné dáta a kontrolný materiál. Pri ich pravidelnom periodickom sledovaní vidí „kde je“ a zbadá, na čo treba dať dôraz, na čo upriamiť pozornosť a aké prípadné zmeny „smeru“ sú potrebné.

„... Ak nový manažér objaví podrobnú, zrozumiteľnú a kritickú povahu dokumentov už na začiatku, začne ich považovať skôr za priateľské nástroje než za protivnú ťažkú povinnosť. ‘Nastaví‘ tak svoje smerovanie ráznejšie a rýchlejšie.“ (Výstižné a jasné, bez komentára.)

Kód na správny „kód“ a riadenie (3.05, 3.06)

Spätná kontrola, viditeľnosť procesu, informovanie o jeho stave sú tiež dôležité. Zároveň s vývojom je vhodné publikovať stav produktivity, výskytov chýb, stavu postupu (kde sme, koľko ešte treba urobiť, kde sú problémy), odhadových pravidiel, atď. Zdieľaním takých dát môže profitovať celá vývojová komunita.

Sklz a s ním súvisiaca snaha o záchranu plánu je v softvérovej branži častý jav. Mali by sme mať však na pamäti varovanie: „Pridávanie ľudských síl (programátorov) do oneskoreného softvérového projektu ho môže ešte viac oneskoriť

“. Vyberám pár zaujímavých postrehov k tejto problematike (odhadu počtu „človeko-mesiacov“) z už citovanej knihy:

„Dĺžka projektu v mesiacoch závisí od jeho sekvenčnej povahy (stavby). Maximálny počet ľudí zase závisí od počtu nezávislých podúloh (podproblémov). Z týchto dvoch faktorov možno potom odvodiť len plán použitím menšieho počtu ľudí a viacerých mesiacov… Nedá sa však získať funkčný plán použitím viacerých ľudí a menšieho počtu mesiacov. Mnohé softvérové projekty sa dostali ešte do väčších problémov práve skôr ´zľaknutím sa‘ kalendára než súhrnom iných faktorov.“

Teoretické práce a poznatky v oblasti softvérového inžinierstva poskytujú širokú paletu pohľadov na rôzne typy softvérových procesov, architektonických vzorov, analytických dátových vzorov a iných techník. Profesia softvérového inžiniera si vyžaduje zorientovať sa v problematike rôznych profesionálnych štandardov a teoretických poznatkov.

Validácia, testovanie, servis (3.10, 3.14, 3.15)

Dôležitým prvkom najmä v záverečných fázach tvorby softvérového výrobku je spätná väzba, mechanizmus neustálej kontroly produktu. Toto by mohla zabezpečiť od vývojárov nezávislá testovacia skupina „testerov“. Táto skupina overuje, porovnáva korešpondenciu už naprogramovaného voči špecifikácii, snaží sa odhaliť každý nesúlad, chybu či nejednoznačnosť. Predpokladám tu existenciu väčšej softvérovej firmy. V opačnom prípade by to mal byť aspoň jednotlivec… v extrémnom „mini“ prípade aspoň programátori sami. Dôležité asi je, aby testovaný prvok (funkčnosť, časť, modul, dizajn,…) spätne nekontrolovali len tí istí ľudia, ktorí ho aj „kódovali“. Nenarážam na vedomú tendenciu nechcieť vidieť svoje chyby, ale chýbal by tu potom potrebný zdravý nadhľad, čistota a čerstvosť verifikátorovho pohľadu, hrozí podvedomá snaha vidieť „to“ také, aké to malo byť a nie aké to je.

Myslime na to, že skutočne nezávislým a najnáročnejším „auditorom“ bude napokon sám zákazník. V časovom horizonte pri budúcom používaní programu určite odhalí každý jeho kaz a chybu. Spomínaná testovacia skupina by sa teda mala vžiť do úlohy zákazníka, musí vidieť softvérový produkt jeho kritickými očami a byť motivovaná chyby naozaj nachádzať. Potom sa jej snáď časom podarí objaviť miesta, kde boli zámery návrhárov zle pochopené, alebo nesprávne implementované.

Alternatívnou možnosťou na zabezpečenie účinnej priebežnej validácie vyvíjaného produktu by mohlo byť zriadenie testovacej skupiny priamo na pôde zákazníka a v jeho réžii. Predpokladajme, že vytváraným produktom je informačný systém pre finančnú inštitúciu, teda zákaznícky softvér (vytváraný na zákazku pre jedného zákazníka). Zákazník by mohol mať prirodzený záujem zriadiť nielen tím ľudí zodpovedných za plynulé nasadzovanie novej informačnej technológie, ale hlavne na jej testovanie. Činnosť testovacieho tímu by sa mohla začať, akonáhle by bola hotová prvá „zrelá“ verzia produktu s implementovanými kľúčovými modulmi a

základnou funkčnosťou (predpokladám ešte dlhý nasledujúci vývoj v živom kontakte softvérovej firmy so zákazníkom). Ak by včleňovanie nových komponentov do projektu ale hlavne neustála validácia, testovanie, pripomienkovanie, vyhotovovanie námetových listov, a i. bolo v kompetencii práve vyššie spomínanej testovacej skupiny - zamestnancov zákazníka, bola by zabezpečená integrita a bezpečnosť cenných dát ale hlavne: „ostré oko“ testovacieho tímu.

Dobre fungujúca testovacia skupina, ktorá začne pracovať dostatočne včas a priebežne počas vývoja, je teda určite tiež potrebným mechanizmom na zabezpečenie budúcej plnej spokojnosti koncového zákazníka. Softvéroví inžinieri by mali zabezpečiť a naštartovať tento mechanizmus. Môžeme to považovať za etický princíp, ale i za samozrejmosť, ku ktorej firmu donúti trh, ak chce profitovať a zachovať si dobré meno.

Po oficiálnom nasadení softvéru by mala byť zabezpečená i údržba (prirodzene len ak bude potrebná a žiadaná). Servis by mal byť predsa samozrejmý pre každého seriózneho výrobcu, softvérovú firmu nevynímajúc.

Záver

V eseji som sa určite nedotkol všetkých aspektov súvisiacich s kvalitou softvérového výrobku a jeho výroby. Pokúsil som sa rozviesť niektoré ťažiskové body jedného z princípov tvoriacich kódex etiky softvérového inžinierstva. Nazdávam sa, že ide o najpraktickejšiu časť kódexu a zamýšľať sa nad ňou určite zmysel má.

Použitá literatúra:

[1] Frederick P. Brooks: The Mythical Man-Month. Addison-Wesley.

[2] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

Po prečítaní tejto eseje musel Wesly uznať, že už v roku 2000 boli značne predvídaví. Dodržiavanie štandardov bolo naozaj obrovským problémom. Dnes už samotné dodržiavanie problematické nebolo. Centrálne celosvetové kontroly vedeli odhaliť akúkoľvek chybu v štandarde vytvoreného softvéru. Problematickým je dnes skôr určenie a vytvorenie štandardov, ktoré by zabezpečili etickú, ľudskú či ekologickú kvalitu novovytvoreného softvéru. Dnes je vývoj taký rýchly, že výbor určujúci softvérové štandardy BSS (Board of Software Standards) je v plnej pohotovosti 24 hodín denne.

Spomenul si na dobu, keď sa ešte venoval programovaniu. Nevedel síce ako sa to robí dnes, ale za jeho éry sa každý kód musel pred vypustením skontrolovať v GSTC (global sofware testing centre). GSTC bol vlastne jeden obrovský počítač, na ktorý sa programátor pripojil vždy, keď otvorili svoje programátorské prostredie. Akýkoľvek pokus o kompiláciu alebo vypustenie kódu znamenal, že GSTC prezrel kód a upozornil programátora na porušenie akejkoľvek normy. Niekedy to Weslymu liezlo na nervy. Hlavne vtedy keď zabudol dôkladne okomentovať kód a GSTC ho prinútilo aby tam komentáre doložil. Nie celkom si vedel predstaviť ako to mohlo fungovať v dobe pred GSTC. Musel ale uznať, že práve táto nepriama kontrola programátorov značne uľahčila a sprehľadnila tvorbu softvéru a dodržiavanie kvality je dnes viac či menej samozrejmosťou.

Cukrík a bič

Peter Liczki
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
liczki@decef.elf.stuba.sk

Abstrakt: Softvérové inžinierstvo má oproti iným, klasickým povolaniam nevýhodu v tom, že sa objavilo so značným oneskorením. Preto mnoho vecí, ktoré sú v iných profesiách už dávno vyriešené, sa v softvérovom inžinierstve riešia iba teraz. Takou je aj otázka profesionálnej etiky a istého záväzného etického kódexu. Tento priestor sa snaží vyplniť etický kódex softvérového inžinierstva, ktorý je aj témou pre túto esej. V nej sa bližšie pozrieme na bod týkajúci sa profesionálnej rozvahy softvérového inžiniera.

Etický kódex softvérového inžinierstva ACM a IEEE, 4. princíp: Rozvaha [1]

Softvéroví inžinieri sa usilujú o integritu a nezávislosť pri svojom profesionálnom rozhodovaní. Presnejšie, softvéroví inžinieri by mali dodržovať nasledovné body:

4.01. Prispôsobiť všetky svoje technické rozhodnutia potrebe podporovania a chránenia ľudských hodnôt.
4.02. Hlásiť sa iba k takým dokumentom, ktoré boli vyhotovené pod ich riadením alebo patria do oblasti ich kompetencií, s ktorými sú v zhode.
4.03. Snažiť sa o profesionálnu objektivitu pri posudzovaní akéhokoľvek softvéru alebo prislúchajúcej dokumentácie.
4.04. Nezúčastňovať sa na žiadnych protizákonných finančných praktikách ako úplatkárstvo, dvojité účtovanie alebo iných finančných podvodoch.
4.05. Oboznámiť všetky zainteresované strany s vyskytnutým konfliktom záujmov, ktorému sa nedalo vyhnúť ani zabrániť.
4.06. Odmietnuť účasť ako členovia alebo poradcovia v privátnych, štátnych, alebo profesijných zboroch zaoberajúcich sa záležitosťami týkajúcich sa softvéru, v ktorých oni, ich zamestnávatelia alebo klienti majú nepopierateľné potenciálne konflikty záujmu.

Softvéroví inžinieri sa počas svojej profesionálnej kariéry ocitnú mnohokrát v situáciách, kedy musia zaujať jasné stanovisko a priniesť ťažké rozhodnutia etického charakteru. Nejedná sa však o jednoduché rozhodnutia medzi bielym a čiernym, dobrým a zlým. Etické kódexy v týchto prípadoch vlastne pomáhajú tým, že

naznačujú, ktorá cesta je z etického hľadiska tá správna a tým často uľahčia rozhodovanie. Problémom rozvahy a etického rozhodovania sa zaoberá aj jeden bod etického kódexu softvérového inžinierstva.

Ľudské hodnoty verzus zamestnanie

Z pohľadu človeka, ktorý musí rozhodnúť medzi tým čo je dobré pre neho a čo pre celú spoločnosť, je to často výberom na základe vlastného svedomia. Bod 4.01 etického kódexu softvérového inžinierstva napovedá, že v takýchto prípadoch je etické to rozhodnutie, ktoré je dobré pre spoločnosť a ktoré je v súlade s ľudskými hodnotami. Etický kódex vlastne chráni ľudskú spoločnosť (všeobecné dobro) a záujmy jednotlivcov sú v podstate zaručené tým, že sa u každého predpokladá dodržiavanie kódexu. Nie vždy je však tomu tak. Presne do takejto nepríjemnej situácie sa dostala aj testerka Cindy Yardley zo spoločnosti Silicon Techtronics z knihy The Case of Killer Robot od autora Richarda G. Epsteina [2]. Jedného dňa ju jej šéf Ray Johnson požiadal o to, aby vykonala podvod. Požiadal ju, aby sfalšovala testy, čím zatajila skutočnosť, že program vyvinutý v ich spoločnosti na ovládanie robota nefunguje korektne, čo v konečnom dôsledku spôsobilo smrť jedného človeka. Cindy Yardley by bola mohla povedať nie, bola by sa tak zachovala eticky a možnože by bola zachránila život jedného človeka. Bol v tom však háčik, jej šéf Ray Johnson nechcel, aby sa takto rozhodla, a tak sa jej vyhrážal, že týmto prístupom môže prísť o prácu: „Povedal mi, že ak nesfalšujem výsledok testu, tak všetci v divízii stratia svoju prácu.“ Naopak prvý výber sa snažil zatraktívniť tým, že jej to môže pomôcť v budovaní jej kariéry. Jedným slovom jej ponúkal cukrík, pokiaľ mu vyhovie a bude sa správať neeticky, na druhej strane sa jej vyhrážal bičom, ak bude počúvať svoje svedomie.

V podstate mala na výber tri možnosti. Mohla to odmietnuť a o celej veci informovať vedenie spoločnosti a prípadne verejnosť, mohla to odmietnuť a o celej veci mlčať (poprípade svojmu šéfovi odporučiť iného testera, teda filozofiou „ja si umývam ruky, ja s tým nič nemám“) a nakoniec mohla to aj prijať a zúčastniť sa na podvode. V prvom prípade by možno bola mohla zabrániť smrti jedného človeka, bolo by to ale znamenalo koniec jej profesionálnej kariéry. V Silicon Techtronics by istotne neboli tolerovali jej nelojálnosť k svojmu nadriadenému, alebo dokonca k celému podniku. Ona si vybrala tretiu možnosť, čo viedlo k tomu, že sa stala spoluvinníkom na smrti mladého testera, ktorého zabil robot ovládaný chybným softvérom. V danej situácii, keď ešte nepoznala možné následky, bola k dispozícii aspoň možnosť druhá. Možnože by si zachránila svoje zamestnanie, ale aj tak by niesla spoluzodpovednosť za smrť človeka – možno nie pred verejnosťou, no pred svojím svedomím istotne. Ona sa jednoducho nemohla rozhodnúť zo svojho pohľadu správne, sama nemohla vyriešiť túto situáciu. Okolie jej nedalo na výber. Ak okolie nedodržuje isté etické normy, ostáva buď odísť, alebo sa správať rovnako.

Vráťme sa však späť k etickému kódexu a to k bodu 4, ktorý sa zaoberá otázkou profesionálneho prístupu v rozhodovaní.

Profesionalita verzus sympatie

Bod 4.3 hovorí o tom, že softvérový inžinier, pokiaľ ide o jeho profesionálnu prácu, by mal zabudnúť na svoje súkromie, v zmysle takom, že pri hodnotení práce iných by nemal brať ohľad na to, kto to robil a aké sú ich vzťahy v súkromí. Nie vždy je to však samozrejme ľahké, ale pri svojom hodnotení by mal vlastne vychádzať iba z práce, ktorú má pred sebou a z vlastných profesionálnych skúseností. Ak sa bude snažiť vo svojich pripomienkach a hodnotení o úplnú objektivitu a konštruktívnosť, zmenší sa tým aj možnosť toho, že autor daného softvéru si jeho pripomienky vysvetlí ako útok na jeho vlastnú osobu a celý problém sa prenesie z odbornej roviny do úplne inej, nežiaducej.

Bod 4.4 sa zaoberá finančnou bezúhonnosťou softvérového inžiniera. Jedná sa o úplatkárstvo a finančné podvody. V dnešnej dobe je úplatkárstvo veľkým problémom vo všetkých oblastiach života. Je problémom napríklad konkurzných konaní, kde sa na istú prácu vyberá spoločnosť na základe ponuky riešenia. V týchto prípadoch je zvykom zo strany uchádzačov prejaviť istú priazeň komisii, ktorá rozhoduje o víťazovi konkurzu, menším darčekom. Je to v poriadku, kým sa to nachádza v istých rozumných medziach. Považuje sa to iba za istú pozornosť zo strany danej spoločnosti. Treba však presne poznať hranicu medzi nevinnou pozornosťou a úplatkárstvom. Softvérový inžinier či už v role uchádzača alebo hodnotiaceho si musí byť vedomý, pokiaľ môže zájsť, aby si to druhá strana nevysvetlila zle alebo aby to výraznejšie neovplyvnilo konečné rozhodnutie. Otázka, kde je táto hranica, je namieste, ale ťažké je zodpovedať ju. Neuvediem tu priamu odpoveď, avšak ako merítko môže pomôcť, akú pozornosť by som dal, keby to malo byť verejne známe.

Konflikt záujmov

Etický kódex sa dosť výrazne zaoberá aj otázkou konfliktu záujmov. Je to veľmi dôležitý aspekt profesionálneho správania. Hrať na dve strany tak, aby som z toho mal čo najviac ja, je neetické a nie je hodné žiadneho profesionála vo svojom odbore.

Veľmi pekný príklad konfliktu záujmu si môžeme prečítať v už spomínanej knižke The Case of Killer Robot od Richarda G. Epsteina [2]. V nej sa John Cramer, vedúci projektu výroby softvéru na ovládanie robota, dostal do situácie, keď mohol svoje postavenie zneužiť a vyťažiť z nej čo najväčší zisk pre seba.

Popri tom, že pán Cramer bol vedúcim projektu v Silicon Techtronics, bol aj hlavným akcionárom v istej softvérovej firme Lucrative, ktorá sa zaoberala predajom a zavádzaním vývojárskeho softvéru. Táto firma sa začala zaoberať vývojovými prostriedkami na objektovo orientované techniky špecifikácie, návrhu a implementácie, čo bolo v tej dobe úplne novým prístupom k vývoju softvéru. J. Cramer začal presadzovať v Silicon Techtronics, aby sa v projekte použila táto nová, v tej dobe ešte veľmi nová, v praxi málo používaná a teda ešte nie celkom otestovaná paradigma programovania. Pre firmu Lucrative tak vybavil celkom pekný obchod tým, že sa stali dodávateľmi CASE nástrojov a poskytovateľmi poradenských a školiacich služieb pre Silicon Techtronics.

Možno to pán Cramer tak nemyslel, možno to myslel naozaj dobre. Možno naozaj videl budúcnosť v tomto novom prístupe k vývoju softvérového systému a možno poznajúc jej výhody a prednosti chcel naozaj len dobre. Avšak spravil dve veľké chyby.

Značne ovplyvnil rozhodovanie ohľadom zmeny a pri výbere dodávateľa v danej veci i keď vedel, že mu pre jeho vzťah s firmou Lucrative vznikol konflikt záujmov a neupozornil vedenie spoločnosti Silicon Techtronics na svoju previazanosť s firmou Lucrative. Ba čo viac, zamestnanci neboli spokojní ani s poskytovaným softvérovým vybavením a ani so službami firmy Lucrative. Brzdilo to vývoj celého systému a programátori boli pod neustálym stresom kvôli dodržaniu stanovených termínov. Avšak na základe uzavretej zmluvy nemusela firma Lucrative niesť žiadnu zodpovednosť za zistené nedostatky.

Za tieto skutočnosti nesie plnú zodpovednosť pán John Cramer a tým sa jeho meno dostalo na zoznam ľudí zodpovedných za smrť mladého človeka. Myslím, že niet pochýb o tom, či sa pán Cramer zachoval eticky, samozrejme sa tak nezachoval. Ako ináč sa však mohol rozhodnúť? Mohol navrhnúť možnosť prechodu na tento nový prístup vo vývoji softvéru a ďalej sa do debaty nezapájať, nanajvýš ak informatívne o tejto novej paradigme programovania. Mohol informovať vedenie Silicon Techtronics a všetkých zainteresovaných o svojich vzťahoch s firmou Lucrative a stať sa aktérom v debate v tomto svetle. Hociktoré z týchto dvoch možností vyriešenia danej situácie by zodpovedalo viac duchu etického kódexu softvérového inžinierstva ako to, čo si zvolil pán Cramer.

Dnes je to už nepopierateľným faktom, že aj softvérové inžinierstvo dozrelo do štádia, kedy je potrebné zaviesť isté etické normy. Je však problémom, že neexistuje žiadna organizácia, ktorá by združovala všetkých softvérových inžinierov. V podstate pre tých softvérových inžinierov, ktorí nie sú členmi IEEE alebo ACM, body hore citovaného etického kódexu softvérového inžinierstva nie sú záväzné. Do tej doby, kým nevznikne takáto veľká organizácia združujúca všetkých softvérových inžinierov, by bolo vhodné etické otázky riešiť na úrovni podnikov a firiem. Tieto zmeny by mali byť iniciované z vedenia podnikov a nastolené etické pravidlá by mali platiť bez výnimky pre každého. Vhodným základom pre takýto kódex by mohol byť práve etický kódex softvérového inžinierstva od IEEE Computer Society a ACM.

Použitá literatúra:

[1] Gotterbarn D., Miller K., Rogerson S.: Software Engineering Code of Ethic and Professional Practice, IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practise, October 1999, IEEE Computer, str. 84-88

[2] Richard G. Epstein: The case of the killer robot. John Willey & Sons, 1997. (ISBN 0-471-13823-1)

Tento princíp sa celkom netýkal iba softvérovej problematiky, napadlo Weslyho. Korupcia, úplatkárstvo a iná forma nekalej činnosti sa predsa vyskytujú v každej ľudskej profesii. V tejto oblasti sa toho za 50 rokov veľa neudialo, lebo finančné podvody a podobné aktivity sú bežné dodnes aj keď možno v inej, oveľa sofistikovanejšej podobe. Veď byť dnes hacker je skôr profesia ako nevinná forma zábavy. Dnešní hackeri sú členmi rôznych nezávislých skupín, ktoré sa živia práve finančnými podvodmi a inými nepríjemnosťami. Aj keď sú aj takí, ktorých najmú bezpečnostné agentúry, aby sa snažili útokom a podvodom predísť. Hackeri sú silnou komunitou a mnohokrát veľmi nebezpečnou pre spoločnosť. Veď len nedávno sa skončila 4. elektronická vojna medzi hackermi a čínskofilipínskou vládou. Hackeri počas tejto vojny paralyzovali cestnú dopravu, spôsobili výpadky prúdu a aj ďalšie problémy. Softvérová polícia si s nimi nevie rady a mnohí extrémisticky zmýšľajúci ľudia chcú, aby boli proti nim nasadené radikálne opatrenia.

Weslyho ešte na eseji zaujalo, že autor cituje knihu The Killer Robot case, ktorá bola počas jeho štúdií veľmi často spomínaná, ako priekopnícke dielo v oblasti etiky.

Napadlo mu, že skočí pozrieť manželku domov a tak si zapol holoteleport. Znova mu napadlo, že by si už konečne mohol zmeniť svoj profil v ňom, ale nakoniec to neurobil, lebo sa mu nechcelo. Na obrazovke si ešte pripravil ďalšiu esej a za malú chvíľku už sedel doma.

Manažment vývoja softvérového produktu

Ján Glončák
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, Bratislava 831 02
junak@junak.sk

Abstrakt: Manažment v softvérovom inžinierstve je jedna z najdôležitejších častí pri vývoji softvérového produktu. Základnou úlohou dobrého manažéra pri každom projekte je zabezpečiť realistický odhad nákladov, personálu, stanoviť mieru neistoty a riziká projektu. Na základe morálneho a etického prístupu sa snažiť správnou motiváciou a organizáciou práce softvérových inžinierov o zvyšovanie kvality ponúkaných softvérových produktov. Manažér je zodpovedný nielen za kvalitu výsledného softvérového produktu, ale aj za vykonanú prácu celého vývojového tímu a dodanie softvéru načas. V prvej časti tejto eseje vyjadrím svoj názor na to, aké sú princípy dobrého manažmentu projektu vrátane odhadu rizík a problémov s tým súvisiacich. V druhej časti naznačím, aké je pre úspešnosť projektu dôležité dbať na bezpečnosť údajov a informačných zdrojov.

Etický kódex softvérového inžinierstva ACM a IEEE, 5. princíp: Manažment

Manažéri softvérového inžinierstva by mali podporovať etický prístup k manažmentu vývoja a údržby softvéru. Presnejšie povedané, mali by sa riadiť nasledovnými pravidlami:[1]

5.01 Zaistiť dobrý manažment pre každý projekt, vrátane efektívnych pracovných postupov na zvyšovanie kvality a znižovanie rizika.
5.02 Uistiť sa, že sú softvéroví inžinieri informovaní o štandardoch predtým, ako sa vyžaduje ich dodržiavanie.
5.03 Uistiť sa, že softvéroví inžinieri poznajú pravidlá a postupy na ochranu hesiel, súborov a iných dôverných informácií pred ostatnými zamestnancami alebo inými ľuďmi.
5.04 Prideľovať prácu len po zvážení vhodného podielu vzdelania a skúseností so záujmom o ďalšie vzdelávanie a získavanie nových skúseností.
5.05 Zabezpečiť realistický kvantitatívny odhad nákladov, rozvrhu, personálu, kvality a výsledkov projektu, na ktorom pracujú alebo zamýšľajú pracovať, a odhadnúť neurčitosť týchto odhadov.
5.06 Vzbudzovať záujem potenciálnych softvérových inžinierov len úplným a presným popisom podmienok zamestnania.

5.07 Ponúkať čestné a spravodlivé odmeňovanie.
5.08 Nebrániť niekomu, aby zaujal pracovnú pozíciu, ak je na ňu primerane kvalifikovaný.
5.09 Zabezpečiť čestnú dohodu o vlastníctve softvéru, metód, vývoja a iného intelektuálneho vlastníctva, na ktorom sa podieľa softvérový inžinier.
5.10 Vytvoriť spôsob postihov v prípade porušovania predpisov zamestnávateľa alebo tohto etického kódexu.
5.11 Nežiadať softvérového inžiniera, aby urobil niečo, čo by nebolo v súlade s týmto etickým kódexom.
5.12 Netrestať nikoho za vyjadrovanie etických zásad pri realizácií projektu.

„Napriek ekonomickej a funkčnej dôležitosti softvéru sa vývoj softvéru považuje vo všeobecnosti za veľmi rizikovú oblasť, v ktorej pokrok v manažmente rizík nie je až taký viditeľný ako v ostatných disciplínach. Jedným príkladom ilustrujúcim nedostatočný manažment rizík v softvérovom vývoji je to, že drvivá väčšina problémov softvérových produktov, ktoré nie sú dodané načas, je úzko spätá s problémami softvérových technológií (ako napríklad vývoj zložitých algoritmov), nad ktorými nie je možné striktným spôsobom zabezpečiť kontrolu. Výskumy však popierajú toto tvrdenie a ukazujú, že 45% zo všetkých prípadov omeškania sú problémy úzko súvisiace s organizáciou práce.“

Dale Walter Karolak [2]

Úvod

V realite súčasného sveta sa stretávame s množstvom neúspechov softvérových firiem pri snahe dodať softvérový produkt v dohodnutom termíne. Tento jav sa nevyskytuje len u menších spoločností, pri ktorých by sme mohli namietať, že firma má neschopný manažment a topí sa v množstve organizačných problémov. Možno ho pozorovať aj pri veľkých spoločnostiach, akou je napr. firma Microsoft, ktorá je na čele rebríčka vo vývoji a hlavne predaji softvéru na svete. Práve príklad takýchto veľkých softvérových spoločností a ich manažmentov zaoberajúcich sa softvérovým vývojom nám priam núka hypotézu, že vývoj softvéru je nekontrolovateľná aktivita, ktorá nebude nikdy zvládnutá prijateľným spôsobom. Čo to znamená nezvládnuť proces manažmentu vývoja softvérového produktu? Znamená to dopustiť sa chyby alebo viacerých chýb niekde pri plánovaní nákladov, ľudských zdrojov, čo budú na projekte pracovať, alebo v neposlednom rade zlého odhadu času trvania jednotlivých na seba nadväzujúcich úloh. Tieto chyby môžu mať nepriaznivý ekonomický dopad, čo predstavuje neúspech pri návrate investícií, resp. nespokojnosť zákazníka alebo môžu znemožniť použiteľnosť produktu a jeho možnú podporu v budúcnosti.

Softvérový manažment a odhad rizík vývoja softvérového produktu

Na softvérový manažment sa možno pozerať z dvoch hľadísk: priemyselného a praktického [2]. Priemyselné hľadisko poskytuje obraz o celkovom úsilí softvérového vývoja a rizík spojených s vývojom softvérového produktu. Praktické hľadisko poukazuje na to, čo treba spraviť, prípadne obmedziť v procese softvérového vývoja, aby sa zvýšila pravdepodobnosť úspešnej realizácie projektu. Obidve hľadiská navzájom pomerne úzko súvisia, a preto sa nebudem snažiť nejakým násilným spôsobom definovať, ktorý problém sa týka priemyselného a ktorý naopak praktického hľadiska.

Hoci až 90% všetkých nákladov na počítačové systémy je úzko spätých so softvérom, výroba softvéru nie je často považovaná za samostatnú a hlavnú vývojovú činnosť. Väčšina softvérových spoločností zameraných na poskytovanie riešení na špeciálnej hardvérovej platforme dodáva softvérový produkt spolu s hardvérovým ako koncový produkt a následne nie je veľmi jednoduché stanoviť výšku ceny softvéru. Oneskorenie takéhoto projektu má najčastejšie príčiny práve v tom, že meškú softvérový vývoj. Manažéri softvérového vývoja často tvrdia, že je to pretože softvérový vývoj je natoľko kreatívny, že sa nedá opísať ako ostatné disciplíny. A pretože etapy vývoja softvérového produktu nie sú jednoducho merateľné, aj napriek tomu, že je všeobecne známe, že vývoj softvéru obsahuje riziko, nedá sa vraj toho spraviť veľa, čím by sa riziko neúspechu projektu minimalizovalo.

V nasledujúcich odsekoch sa pokúsim poukázať na niekoľko hlavných problémov, ktoré ako podstatné vyzdvihol Dale Karolak v publikácií „Softvérové inžinierstvo manažmentu rizík“.

Nedostatok pochopenia a výučby o tom, čo je zahrnuté do softvérového manažmentu. Koncepcia manažmentu softvérového vývoja, ktorý obvykle obsahuje istú mieru rizika, sa nedostatočne vyučuje, čo potom v praxi znamená, že je tento jav veľmi podceňovaný. Podceňovanie faktu existencie rizík vedie k závažným pracovným problémom, hlavne pri finalizácii projektu a spôsobuje oneskorené dokončenie softvérového produktu a v konečnom dôsledku stratu pre softvérovú spoločnosť.

Softvéroví inžinieri, manažéri alebo vedúci jednotlivých vývojárskych tímov neprejavujú dostatočnú disciplínu pri implementácií užitočných techník manažmentu. Dokonca aj keď sa nájdu takí, ktorí sú vzdelaní v oblasti softvérového manažmentu a techník, nevenujú potrebné množstvo času a úsilia na pokiaľ možno čo najdetailnejšiu identifikáciu, určenie, kalkuláciu, plánovanie, zhromažďovanie a referovanie softvérových rizík. Som si vedomý, že je veľmi namáhavé a prácne vykonať všetky uvedené činnosti, hlavne ak je softvérový manažér obmedzený rozpočtom a je pod tlakom termínu a časového plánu, ale správny manažér by si mal uvedomiť, že jeho prístup k identifikácií rizík mu umožní lepší odhad celkových nákladov, úsilia a iných prostriedkov na realizáciu softvérového projektu.

Nedostatok potrebných prostriedkov na realizáciu softvérového manažmentu na znižovanie rizík. Aj keď je situácia na trhu vcelku dobrá a nájdeme tam množstvo aplikácií na podporu manažmentu softvérových rizík, je komplikované vybrať si. Softvéroví manažéri by pri výbere takýchto prostriedkov mali brať do úvahy fakt, že softvér na podporu techník na rozpoznávanie rizík musí byť dostatočne jednoduchý a pohodlný, aby ľudia, ktorí s ním budú pracovať, nemali problém s jeho uvedením do praxe, a aby im zjednodušoval a uľahčoval prácu. Inak sa softvér nebude používať a jeho nákup budú len zbytočne vyhodené peniaze.

Obmedzený alebo nedostatočný pohľad na riziká softvérového inžinierstva a zlyhanie pri integrácií činností obsahujúcich riziko. Tento faktor považujem na základe vlastnej skúsenosti za najdôležitejší a najzávažnejší pri manažmente rizík. Pokúsim sa na príklade objasniť z čoho vychádza moje tvrdenie. Istý poskytovateľ internetových služieb v rámci integrácie do nadnárodnej spoločnosti musel zmeniť informačný systém evidencie klientov a vedenia účtovníctva a tak sa prispôsobiť štandardom spoločnosti. Dôvody nutnosti tejto zmeny tu nebudem rozoberať, ale uznávam, že to bolo nevyhnutné, z dôvodu plánov spoločnosti do budúcnosti a potenciálnej veľkosti klientely. Zmena informačného systému nie je samozrejme triviálna záležitosť, ak si uvedomíte, že počet klientov bol niekoľko tisíc a že zmena sa mala uskutočniť bez prerušenia poskytovaných služieb. Aj keď práce začali prebiehať niekoľko mesiacov pred samotným „ostrým“ nasadením nového systému do prevádzky, bolo tu isté riziko, že pri testovaní sa nepodarilo odhaliť všetky chyby. Keďže si spoločnosť nemohla dovoliť poškodiť svoje meno a rátala s určitými problémami, oznámila klientom, že v istom časovom úseku sa môžu vyskytnúť malé problémy, ale požiadala všetkých klientov, aby boli trpezliví, že je to kvôli zvyšovaniu kvality poskytovaných služieb. Manažment softvérového vývoja na základe analýzy problémov s prechodom na nový systém vypracoval časový plán, ktorý presne definoval postup prác pri zmene informačného systému. Plán sa zdal veľmi dobrý a nik nepredpokladal, že sa môže oneskoriť. Všetci si želali, aby celá operácia mala hladký priebeh a išla podľa plánu, pretože inak to mohlo mať katastrofálny dopad na celú spoločnosť a jej postavenie na trhu. Realita však bola iná. Prechod systému sa oneskoril takmer o dvojnásobok času, aký bol avizovaný. Veľká väčšina klientov bola nespokojná a sťažovala sa na kvalitu poskytovaných služieb. Oddelenie styku so zákazníkom bolo permanentne preťažené. A kde sa stala chyba? Prečo zlyhal zdanlivo perfektný plán, ktorý mal dokonca aj nejakú časovú rezervu?

Zásadná chyba bola, že manažér projektu dobre neodhadol čas trvania integrácie jednotlivých častí a testovaciu fázu. Druhá chyba bola, že bolo nutné spraviť priveľa úkonov, ktoré boli od seba závislé, v krátkom čase a pri časovom sklze jednotlivca začal meškať celý vývojový tím. Nechcem teraz polemizovať o tom, či to bola chyba jednotlivca alebo nie, ale jedno je isté, projektový manažér si pri realizácií plánu projektu neuvedomil, že musí brať do úvahy vývoj celého životného cyklu. Mal na vec celkom zjednodušený pohľad, pretože podcenil fázu integrácie jednotlivých častí a testovaciu fázu. Z uvedeného príkladu jednoznačne vyplýva, že skutočne efektívny a presný pracovný plán a manažment rizík vývoja

softvéru je možné spraviť len v prípade, že sa na riešený problém pozeráme z globálu a berieme do úvahy celý životný cyklus vývoja softvéru.

Úspešnosť softvérového projektu úzko súvisí aj s tým, ako zodpovedne pristupujú softvéroví inžinieri k ochrane údajov a informačných zdrojov. Zachovanie diskrétnosti a ochrana údajov často nie je len otázka etického konania softvérového inžiniera, ale môže majetkovo poškodiť softvérovú spoločnosť alebo iný subjekt. Preto je veľmi dôležité, aby softvéroví inžinieri boli oboznámení s bezpečnostnými pokynmi na ochranu údajov a dodržiavali ich.

Ochrana údajov a informačných zdrojov

Tým, že ľudia vyvinuli počítačové aplikácie a začali ich používať na spracovanie údajov, sa v značnej miere zvýšila naša pohotovosť a efektívnosť vykonávania jednotlivých úloh. Pri dnešnej technológií počítačov je veľké množstvo údajov sústredené na jednom mieste. Okrem množstva výhod, ktoré priniesla technológia v hromadnom spracovaní údajov, prináša so sebou aj nemalé problémy týkajúce sa miery bezpečností spracovávaných údajov. [3]

Hoci sa v počítačovej technológií dosiahol nesmierny pokrok, veľmi málo sa robí v oblasti osvety používateľov pri ochrane dôverných údajov a informácií pred neautorizovanou zmenou, prezradením, či zničením. Pokúsim v krátkosti naznačiť, čo odporúčam každému manažérovi alebo vedúcemu tímu prízvukovať svojim podriadeným, aby neohrozil výsledok projektu zlými bezpečnostnými opatreniami.

Ktoré údaje sú dôležité, a ktoré treba chrániť?

Všetky údaje sú do určitej miery dôležité. Konkrétnejšie povedané v manažérskych informačných systémoch môže spôsobiť nesprávna alebo neplatná informácia nesprávne manažérske rozhodnutie, ktoré môže vyústiť do straty finančných prostriedkov, energie a času ľudí pracujúcich na projekte. Na druhej strane niektoré údaje sú veľmi citlivé na nečestnú manipuláciu, pretože môžu byť zneužité za účelom neoprávneného získania cudzieho majetku a spôsobenie škody inému subjektu. Najjednoduchší spôsob, akým sa dá vyjadriť dôležitosť a hodnota údajov, je položiť si otázku: „Čo by sa stalo, keby sa tieto údaje poškodili, stratili alebo dostali do rúk neoprávneným osobám?“ Ak si zodpovedne odpoviete na túto otázku, budete sa vedieť rozhodnúť, aký veľký dôraz treba klásť na zabezpečenie údajov.

Dôležité zásady bezpečnosti sa budem snažiť vysvetliť na príklade práce v tíme, teda ak v nasledujúcom texte označím niekoho ako člena tímu, tak dotyčný predstavuje osobu, ktorá má prístup k dôverným údajom.

Základné pravidlo na zachovanie bezpečnosti je, čo najpresnejšie definovať povinnosti všetkých osôb, ktoré prichádzajú do styku z dôvernými informáciami a presne si stanoviť aj spôsob postihovania za nedôslednosť či nedbalosť pri dodržiavaní bezpečnostných pravidiel v zmysle bodu 5.10. etického kódexu. Aj keď sa to zdá pomerne jednoduché, v praxi nie je vždy možné presne definovať mieru zodpovednosti jednotlivých členov tímu za bezpečnosť údajov, pretože vo veľa prípadoch má prístup k dôverným údajom viacero osôb naraz. Manažér musí viesť všetkých členov tímu k dodržiavaniu bezpečnostných zásad a musí dbať o to, aby sa všetci riadili heslom: „Za bezpečnosť je zodpovedný každý!“. Znamená to, že každý, kto má prístup k dôverným údajom, by si mal uvedomiť možné následky a dôsledky svojho správania.

Na to, aby sa v softvérovej spoločnosti alebo v akejkoľvek inej firme dodržiavali isté bezpečnostné pravidlá, je nutné, aby bola presne určená jedna osoba alebo skupina osôb, zodpovedná za zabezpečenie ochrany údajov a definovanie práv jednotlivých používateľov. Túto osobu budem ďalej nazývať supervízor. Supervízor je akýsi dozorca alebo strážca informácií. Má prehľad o dôležitosti ochrany údajov a musí zabezpečiť, aby všetci používatelia systému alebo osoby, ktoré prichádzajú do styku s dôvernými údajmi, boli zaškolené riadiť sa bezpečnostnými postupmi, čo je spomenuté v bode 5.3. etického kódexu.

Softvérový manažér, ale musí rátať s možnosťou, že ani pri používaní akýchkoľvek bezpečnostných metód alebo spôsobu ochrany údajov nikdy nie je schopný zaručiť absolútne zabezpečenie dôverných údajov. Dôležité je vysvetliť všetkým členom tímu, akú úlohu v bezpečnosti hrá ľudský faktor, a že bezpečnosť je dôležitá na každej úrovni. To znamená, aby bezpečnosť nepodceňoval ani ten, kto nepracuje priamo s dôvernými údajmi. Pretože získanie práve jeho hesla môže byť kritickým faktorom pre neoprávnený prístup k údajom. Je dôležité, aby každý člen tímu poznal základné pravidlá pre ochranu hesiel, ktoré sa dajú v krátkosti zhrnúť do nasledovných bodov:

- výber dobrého hesla, najlepšie kombinácia alfanumerických a interpunkčných znakov,
- pravidelná zmena hesla, resp. zmena hesla v prípade podozrenia, že ho vie niekto iný,
- nikdy heslo nezverejňovať, nikomu ho neprezradiť ani nepísať na papier.

Napriek tomu, že spomenuté zásady sú všeobecne známe, ich používanie v reálnom živote je často žiaľ len želanie bezpečnostných pracovníkov. Je nutné, aby supervízor vytvoril prostredie, ktoré bude istým spôsobom nútiť používateľa dodržiavať bezpečnostné pravidlá tým, že mu napríklad nedovolí vytvoriť heslo kratšie ako istý počet znakov, ktoré musí obsahovať alfanumerické a zároveň aj interpunkčné znaky a nastaví dátum konca platnosti hesla atď.

To bolo niečo k zásadám bezpečnosti členov tímu, ale na čo netreba zabudnúť, je ešte jeden problém, ktorý považujem za veľmi dôležitý pri zachovávaní bezpečnosti údajov a informačných zdrojov. V oblasti vývoja softvéru je veľmi častým javom zmena povolania a prechod napríklad ku konkurencii, pretože softvérový inžinier vo väčšine prípadov aj pri zmene zamestnávateľa ostane pracovať v rovnakej oblasti softvérového inžinierstva. Aby nedošlo k zneužitiu údajov osobou, ktorá k nim mala v minulosti prístup, je nutné po rozviazaní pracovného pomeru s každým pracovníkom zrušiť jeho prístup do systému a k informačným zdrojom. Čo však robiť v prípade, že zamestnanec, povedzme bývalý správca, ktorý odišiel, vie o systéme toľko, že nie je možné zamedziť mu prístup k údajom jednoduchým zrušením konta? Tu je situácia podstatne komplikovanejšia, pretože keďže už nie je zamestnanec firmy, nie je povinný dodržiavať bezpečnostné pravidlá určené pre zamestnancov za účelom ochrany údajov. V takomto prípade sa softvérový inžinier len na základe svojho svedomia a morálnych vlastností rozhodne, či bude aj naďalej dodržiavať bezpečnostné zásady alebo nie. Softvérové spoločnosti sa však nemôžu spoliehať na morálne vlastnosti svojich bývalých zamestnancov, a preto sa tieto prípady snažia riešiť pracovnou zmluvou, ktorá zaväzuje osobu, aby aj po rozviazaní pracovného pomeru nezneužila dôverné údaje v neprospech spoločnosti.

Záver

Podľa vlastnej skúsenosti viem, že implementácia manažérskych techník a dôsledných bezpečnostných opatrení do praxe nie je ani zďaleka jednoduchá. Softvérový inžinier, ktorý sa riadi pravidlami etického kódexu a má snahu o zavedenie etických zásad do vývoja softvéru, sa pravidelne stretáva s nevôľou starších a skúsenejších pracovníkov pristúpiť na zásady etického a morálne čistého vývoja softvéru a zmeniť svoje zaužívané a nie vždy korektné zvyklosti. Je to možno aj preto, že komunita ľudí pracujúcich na tvorbe a vývoji softvéru u nás si neuvedomuje, že presným dodržiavaním zásad etického kódexu sa zlepší kvalita ich práce a to so sebou prináša aj možnosť na lepší život. Preto treba vynaložiť ešte veľa úsilia, aby sme tento stav vylepšili. A nám, potenciálnym softvérovým inžinierom, to ponúka veľkú výzvu do budúcnosti.

Použitá literatúra

[1] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[2] Dale Walter Karolak: Software engineering Risk Management. 1995. (ISBN 0-8186-7194-7)

[3] Rebecca Vasvary, Gregg Bass a kol.: Protecting Information Resources. marec 1997.

[4] Mária Bieliková: Manažment v softvérovom inžinierstve. Slovenská technická univerzita, 1999.

Ešte pri čítaní eseje sa mu vybavovali skvelé prednášky profesora Mooda z jeho študentských čias. Mood často prízvukoval, že manažment softvérového projektu je natoľko dôležitý, že by sa mala vytvoriť samostatná profesia, ktorá sa bude zaoberať riadením projektov. Spomenul si, že Moodove slová sa o niekoľko rokov potvrdili a všetky významné univerzity zaviedli manažérstvo softvérových projektov ako samostatný smer v ich výučbovom programe. Rozsiahle výskumy v tom čase ukázali, že zložitosť jednotlivých činností spojených s vývojom softvéru narástla natoľko, že jednotlivec nie je schopný dôkladne sa venovať niekoľkým činnostiam odrazu.

Manažment softvéru sa začal chápať ako samostatná profesia a mnohí významní vedci pracovali na modeloch a metódach správneho vývoja, vedenia a riadenia projektu. Aj on sám pracoval pred niekoľkými rokmi na modeli odhaľovania rizík. V čase dokončenia jeho práce sa však objavil Li Wangov simulačný agent, ktorý vedel s takmer 100% úspešnosťou odhaliť potenciálne riziká softvérového projektu. Li-Wangovi to dokonca vynieslo Nobelovu cenu za softvér a jeho práca bola zakomponovaná do všetkých významných manažérskych nástrojov. Jeho agent bol veľmi zložitý expertný systém, ktorý pracoval s globálnou databázou znalostí. Akýkoľvek problém, či riziko, ktoré sa vyskytlo pri vývoji softvéru, bolo zapísané medzi znalosti a boli pre neho vytvorené náležité spôsoby riešenia a alternatívne postupy. Algoritmus identifikácie rizík nebol doteraz zverejnený, a existujú mnohé dohady, o tom, že Li-Wangovi sa pravdepodobne podarilo vytvoriť umelú inteligenciu 6. generácie. Tak či onak, riziká pri vývoji softvéru boli do značnej miery eliminované. Mnoho manuálnych a opakujúcich sa postupov je dnes prenechaných programovacím robotom a pred programátorov sú postavené len kreatívne úkony. Dnes už nehrozí, že by nebol softvér vytvorený načas, alebo nesprávne. Napriek tomu pribudli problémy ďalšie.

Wesly si ešte prezrel body 5.2 a 5.3 a pomyslel si: „Ach zas tie štandardy“. Nebolo mu celkom jasné, ako sa dalo rozumne programovať bez celosvetových štandardov, ktorých dodržiavanie by bolo dôkladne kontrolované. Nevedel, aké boli na začiatku tisícročia zákony, ale vedel, že dnes je to veľmi prísne. Dodržiavanie štandardu je hlavne v záujme programátora, lebo pri akejkoľvek poruche softvéru môže byť podľa zákonná braný na zodpovednosť.

Profesia alebo mafia?

Marián Varga
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
mvarga@decef.elf.stuba.sk

Abstrakt. Spoločnosť nedôveruje skupine, ktorá má iba technické zameranie, bez kladného a otvoreného vzťahu ku svojmu okoliu. Takýto vzťah musia softvéroví inžinieri vytvoriť pri budovaní svojej profesie. Inak sa môžu nazývať aj mafiou. Esej reagujúca na Etický kódex softvérového inžinierstva organizácií ACM a IEEE, princíp č. 6 – „Profesia".

Etický kódex softvérového inžinierstva ACM a IEEE [1]**, 6. princíp: Profesia**

Softvéroví inžinieri majú podporovať bezúhonnosť a povesť svojej profesie v súlade s verejným záujmom. Obzvlášť majú softvéroví inžinieri primerane:

6.01 pomáhať pri vývoji organizačného prostredia priaznivého pre etické konanie,
6.02 presadzovať, aby softvérové inžinierstvo bolo všeobecne známe,
6.03 rozširovať znalosti v softvérovom inžinierstve vhodnou účasťou v profesných organizáciách, na stretnutiach a publikáciách,
6.04 ako príslušníci jednej profesie podporovať iných softvérových inžinierov v ich snahe nasledovať tento kódex,
6.05 nepodporovať svoj vlastný záujem na úkor profesie, klienta alebo zamestnávateľa,
6.06 riadiť sa všetkými zákonnými ustanoveniami upravujúcimi ich prácu okrem výnimočných situácií, keď ich dodržiavanie nie je v súlade s verejným záujmom,
6.07 byť presní pri oznamovaní charakteristík softvéru, na ktorom pracujú, vyhýbajúc sa nielen nepravdivým tvrdeniam, ale aj tvrdeniam, ktoré možno rozumne považovať za špekulatívne, nič nehovoriace, klamné, mylné alebo pochybné,
6.08 preberať zodpovednosť za objavovanie, opravu a hlásenie chýb v softvéri a v dokumentoch s ním súvisiacich, na ktorých pracujú,
6.09 zabezpečovať, aby klienti, zamestnávatelia a správcovia vedeli o povinnosti softvérového inžiniera dodržiavať tento Etický kódex, ako aj o dôsledkoch tejto povinnosti,

6.10 vyhýbať sa spolupráci s podnikmi a organizáciami, ktoré sú v rozpore s týmto Kódexom,
6.11 uznať, že ak niekto porušuje tento Kódex, je to v rozpore s tým, aby bol profesionálnym softvérovým inžinierom,
6.12 vyjadrovať znepokojenie tým, ktorí sú zaangažovaní vo významných porušeniach tohto Kódexu, okrem prípadov, keď to je nemožné, kontraproduktívne alebo nebezpečné,
6.13 oznamovať významné porušenia tohto Kódexu príslušným orgánom, keď je jasné, že konzultácia s tými, ktorí sú v týchto porušeniach zaangažovaní, je nemožná, kontraproduktívna alebo nebezpečná.

Profesia alebo mafia?

Z Etického kódexu softvérového inžinierstva [1] som sa rozhodol reagovať na princíp „Profesia". Jeho podstatou je, aby skupina ľudí, ktorí sa nazývajú softvérovými inžiniermi, jednak tvorila ***profesiu*** a jednak v očiach verejnosti bola ***bezúhonná, s dobrou povesťou***. Technický pokrok je neutrálny – je rovnako schopný podporovať zlé úmysly, ako dobré. Zo skupiny, ktorá je iba technicky vyspelá a stará sa najmä o svoje vlastné záujmy, sa ľahko môže vykľuť hoci aj ***mafia***. Ak chcú softvéroví inžinieri zabrániť tomu, aby výpočtová technika, ktorej „dušu" tvoria, šírila medzi obyvateľstvom strach a hrôzu, musia byť iní ako mafia.

Princíp „Profesia" je z ôsmich až šiesty v poradí. Predchádza mu päť iných princípov, z ktorých prvý a najdôležitejší je „Verejnosť" [4]. Ak to autori Etického kódexu mysleli vážne, chceli tým povedať, že z dlhodobého hľadiska najlepšou stratégiou, ako získať pre softvérové inžinierstvo dobrú povesť, je byť otvorenými a v prvom rade naozaj slúžiť verejnosti. Nie kamuflovať bezúhonnosť mafiánskym „ja kryjem teba – ty kryješ mňa".

Často sa stretávam s názorom, že jedine v oblasti softvéru je možné za nekresťanské peniaze predať aj najhorší odpad. Málokedy si však ten, kto sa sťažuje, uvedomuje napríklad špecifiká softvéru – jeho obrovskú zložitosť, neviditeľnosť, ba ani to, že práve softvér sa donekonečna prispôsobuje meniacim sa požiadavkám používateľov a rozmanitej ponuke hardvéru. Tento jav je spôsobený nedostatočnou všeobecnou informovanosťou, ***osvetou o softvérovom inžinierstve (bod 6.02)*.**

Softvéroví inžinieri by nemali pôsobiť ako tajná komunita, ktorá dá príučku každému, kto do nej strká nos. Na verejnosť by sa malo dostávať viac ako len používateľské príručky. Azda najlepšie prirovnaním k nástrojom a postupom používaným zavedenejšími profesiami sa dá vysvetliť, aké metódy a techniky softvéroví inžinieri môžu použiť, a či ich aj používajú, aby vytvorili čo najkvalitnejší produkt pri daných zdrojoch a úrovni vedeckého pokroku. Aj pri rýchlom tempe vývoja v našej profesii si jednoducho treba nájsť čas na to, aby sme o tom, čo robíme, zrozumiteľne povedali iným.

Súčasne by verejnosť mala vedieť, že o jej mienku softvéroví inžinieri stoja, a preto ***súhlasia s dodržiavaním Etického kódexu (bod 6.09)***. To neznamená, že sú bez chyby. No mali by mať tendenciu zo zlých skúseností sa poučiť.

Pri tvorbe softvéru majú často softvéroví inžinieri prístup k dôverným informáciám a k manipulácii s cudzím majetkom. Musia odolávať pokušeniu využiť situáciu a ***získať pre seba výhodu na úkor iných (bod 6.05)***. Tiež sa musia vyhýbať konfliktom záujmov, pri ktorých môžu nespravodlivo prihrať výhodu jednej zo strán konfliktu. Je to náročná požiadavka – v záujme pocitu čestnosti (čistého svedomia?) sa vzdať časti svojho ja.

Nemusia ani priamo konať – čestnosť sa môže významne prejaviť už pri ***výrokoch týkajúcich sa softvéru (bod 6.07)***. Kódex usilovne vymenúva viacero druhov tvrdení, ktoré treba zamedziť: „špekulatívne, nič nehovoriace, klamné, mylné alebo pochybné". Dávať si na takéto tvrdenia pozor je veľmi dôležité, pretože polopravda je lož a práve umenie malého klamstva včleneného do pravdivého rámca robí „najúspešnejších" podvodníkov.

Zvlášť dôležitú úlohu hrajú výroky týkajúce sa potenciálnych i skutočných porúch v softvéri. Vtedy mlčanie je najnebezpečnejším klamstvom. Známy je prípad čipu Intel Pentium, ktorý firma bez zábran predávala a propagovala napriek tomu, že jej bola známa chyba tohto čipu [2]. Prípadové štúdie [2] uvádzajú aj prípad firmy, ktorá zatajovala zákazníkom jej známe výskyty porúch v softvéri riadiacom rádioterapeutické zariadenie. Smutným záverom bola smrť dvoch ľudí spôsobená nadmernou dávkou ožiarenia.

Uvedená firma súčasne zanedbávala nielen ***nahlásenie,*** ale aj ***odhalenie a odstránenie*** týchto ***porúch***, takže porušila ***bod 6.08***.

Svetoznámy a ešte stále možno aktuálny je problém roku 2000. Tu akoby softvéroví inžinieri „preetizovali" – v porovnaní s upozorňovaním na veľké nebezpečie potenciálnych zlyhaní výpočtových systémov bol prechod letopočtu nečakane hladký. Na druhej strane veľmi pravdepodobne práve včasné odstránenie chýb, na ktoré sa vyčlenilo dostatok prostriedkov, zabránilo mnohým katastrofám. V každom prípade sa zdá, že toto pozitívne prekvapenie nakoniec nikto neoľutoval, s výnimkou tých, čo chceli (ktovie, či v ich chápaní *eticky*) na Y2K mimoriadne zarobiť.

Schopnosť vytvárať softvér s čo najmenej chybami, vedieť chyby objavovať a odstraňovať neprichádza sama od seba. Nevyhnutné je ***rozširovanie a odovzdávanie znalostí (bod 6.03)***. Mám pocit, že z tohto hľadiska sa nadmerne objavujú niektoré extrémy.

Jednak sú tu *hackeri*, ktorí sledujú všetky detaily, novinky, idú do hĺbky, ale sú zvyčajne úzko zameraní a snažia sa svojimi priam až artistickými trikmi predovšetkým pretromfnúť ostatných. Často sa vyskytujú aj *spiatočníci*, ktorí sú demotivovaní a nie sú ochotní uznať, že v softvérovom inžinierstve sa nedá vystačiť s tým, čo platilo pred 5-10 rokmi.

Treba sa vzdať bojov na vlastnú päsť a dokázať, že aj v oblasti softvérového inžinierstva fungujú normálne medziľudské vzťahy, komunikácia a vzájomná

inšpirácia. V poslednom období v softvérovom inžinierstve cítiť stále intenzívnejší záujem o štandardizáciu. Tá nie je možná bez primeranej diskusie a argumentácie medzi odborníkmi. Neustále konflikty množstva jednostranných, krátkozrako navrhnutých a navzájom nezlučiteľných „štandardov" nerobí tejto profesii dobrú vizitku.

Zvlášť dôležitý je dialóg s inými profesiami – ich skúsenosti a poznatky môžu byť až prekvapivo dobre použiteľné v softvérovom inžinierstve. Naopak, odovzdávaním poznatkov získaných v softvérovom inžinierstve iným profesionálom podporujeme prinajmenšom bod 6.02.

Hovorí sa, že mafia je vždy o krok pred zákonom. V oblasti informatiky žiaľ ***legislatíva (bod 6.06)*** podobne zaostáva za potrebami praxe. Ustanovenia právnych noriem nezodpovedajú súčasnému stavu ani vývoju, nie sú dostatočne konkrétne a na mnohé pálčivé otázky nie sú ani ustálené právne názory.

Len jeden z mnohých – prípad diskusnej služby Prodigy [2] poukazuje na známy problém určenia hraníc slobody prejavu na počítačovej sieti a súčasne zložitý problém zodpovednosti poskytovateľov/výrobcov v prípade neočakávaného/neželaného správania sa produktu/služby.

Preto musia softvéroví inžinieri oveľa častejšie, ako iba vo „výnimočných situáciách" používať vlastný úsudok na to, aby rozhodli, čo je „v súlade s verejným záujmom". Nemali by si to však nechávať pre seba, ale svojou iniciatívou podporiť právnikov ako príslušníkov inej profesie (viď vyššie) v tom, aby vytvorili zákonný rámec, ktorý ukončí stav, keď softvéroví inžinieri musia pracovať v prostredí „divokého východu", oproti ktorému však nie je ani „na západe (takmer) nič nového".

Aj samotný Etický kódex je určitou, aj keď len veľmi voľne formulovanou, normou správania sa. Preto je len prirodzené, ak sa softvéroví inžinieri budú vzájomne podporovať v jeho nasledovaní ***(bod 6.04)***. Súčasne majú softvéroví inžinieri v rámci rozsahu svojej moci klásť odpor voči jednotlivcom, ktorí si myslia, že môžu byť profesionálmi bez etiky ***(bod 6.11)***, voči neetickým organizáciám ***(bod 6.10)***, ale aj aktívne pôsobiť s cieľom nápravy stavu ***(body 6.12, 6.13)***.

Prvý bod ***(6.01)*** na koniec: ***organizáciou*** je aj mafia, nášmu Kódexu by však iste vadilo, že „nie je priaznivým prostredím pre etické konanie". Keď počujeme o *organizovaní*, *organizačnom prostredí* apod., môžeme si predstaviť hierarchické štruktúry, úradníkov, autoritu („krstných otcov"?). Napríklad Tom DeMarco [5] hľadí skepticky na úsilie zaviesť pre softvérových inžinierov oficiálne licencie (udeľované a odnímané vhodnou *organizáciou*) a možno by mal podobný názor aj na akreditácie študijných odborov a certifikácie ich absolventov. Iní [3] ich zase považujú za štandardné prvky ***profesie***, spolu so základným a celoživotným vzdelávaním, profesnými spoločenstvami a, samozrejme, etickým kódexom. Podľa DeMarca softvéroví inžinieri dostávajú licencie každý mesiac na trhu práce a na výplatnej páske. Oficiálne licencie považuje za zbytočnú byrokraciu.

Ja sa domnievam, že organizácia nemusí byť nutne byrokratická. Stačí, ak sa navrhne rozumne, najlepšie s využitím skúseností s organizáciami overenými časom.

Preto by nemali byť „sovietske" (dnes prevláda názor, že socializmus bol *zločinný* režim), ale také, aké sa udržali aj v silne trhovo orientovaných štátoch. Významnú úlohu zrejme hrá otázka informovanosti. Nemohli by licencie alebo niečo podobné udeľovať od štátu nezávislé agentúry, ktoré by tiež navzájom súťažili kvalitou – kvalitou informovania verejnosti a zamestnávateľov o kvalite softvérových inžinierov? To je predsa známa organizácia: samoorganizácia.

Informovanosť má aj iný rozmer: Neviem síce takmer nič o atestáciách lekárov, ale neviem si ani predstaviť, že by dôveryhodný lekár tajil svojmu pacientovi postup liečenia (napríklad s odôvodnením, že má naň autorské práva). Opäť sa vraciam k už spomínanému – je divné, ak najmenej informácií (t.j. dát, ktoré zmenšujú prijímateľovu neurčitosť!) je dostupných práve o informatikoch.

Ťažké je zodpovedať otázku, ako môžeme podporovať etické konanie organizačným prostredím vo vnútri firiem produkujúcich softvér. Už skúmanie, či sa pracuje eticky, prináša podniku dodatočné náklady. Konfrontácia s konkurenciou je boj, v ktorom sa nehľadí na obete. Manažérom ide o splnenie stanovených cieľov, nie o etické názory jednotlivých pracovníkov. Jedine, že by svoje poslanie chápali širšie a tiež prijali Etický kódex za svoj. V tom prípade môže vzniknúť priestor na tímové diskusie o etike alebo na vytvorenie eticky podloženej podnikovej kultúry.

Inak však stále zostáva na verejnosti a zákazníkoch, aby sa informovali podrobnejšie o praktikách týchto firiem a zhodli sa konkrétne na tom, čo treba zakázať.

Quentin Tarantino svojím filmom Pulp Fiction poukázal na to, že aj zločinci majú životné postoje, vyznávajú určité hodnoty a sú to „tiež len ľudia". Tým, že sa stali zločincami, sa ich život nielen nezjednodušil, ale prakticky nemožnou sa pre nich stala aj cesta späť. Pevne dúfam, že tvorcovia softvéru nie sú – ani sa nemusia stať – členmi technicky vyspelej mafie. Ak však majú „rizikové rodinné zázemie" a „nedostatky vo výchove", potom Etický kódex je prvým krôčikom k tomu, aby sa osudu organizovaných zločincov dokázali vyhnúť.

Použitá literatúra

[1] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[2] Spinello, Richard A.: Case Studies in Information and Computer Ethics. Prentice Hall, Upper Saddle River, New Jersey, 1997. ISBN 0-13-533845-X

[3] McConnell, Steve; Tripp, Leonard: Professional Software Engineering: Fact or Fiction? Guest Editors' Introduction. IEEE Software, Vol. 16, No. 6, November/December 1999, str. 13-18.

[4] Gotterbarn, Don: How the New Software Engineering Code of Ethics Affects You. IEEE Software, Vol. 16, No. 6, November/December 1999, str. 58-64.

[5] DeMarco, Tom: Counterpoint: It Ain't Broke, So Don't Fix It. IEEE Software, Vol. 16, No. 6, November/December 1999, str. 67-69.

Weslymu sa táto esej náramne páčila. Uvedomil si, že hoci sa z niekdajšieho softvérového inžinierstva odčlenili samostatné profesie analytikov, architektov, konfigurátorov, komponentárov, je treba priznať, že vo svojich etických kódexoch významne čerpajú z pôvodného Etického kódexu softvérového inžinierstva. Nad niektorými v eseji spomenutými problémami sa musel len pousmiať, najmä nad právnym vákuom, spomínaným v súvislosti s bodom 6.06, hoci treba priznať, že aj pri platnosti Zákona o testovaní softvéru na tradičných platformách a povinnosti denného auditu softvérového procesu sa stále stretávame s obchádzaním týchto predpisov „šikovnými" firmami. Nebolo mu tiež jasné, čo autor vlastne myslel tými „nezávislými agentúrami", snáď len nie organizácie typu Truepeace, ktoré uprednostňujú radikálne akcie napr. za zákaz počítačovej telepatie, odmietajúc rozumné argumenty, namiesto aby prispeli k ďalšiemu zníženiu rizika softvérového zlyhania, ktoré by mohlo poškodiť duševné zdravie asociovaných. Nuž, konfrontácie medzi hospodársky významnými technológiami a etikou sa asi nikdy „nezbavíme".

Wesly bol už dosť vyčerpaný, ale povedal si, že musí tie eseje vybaviť ešte dnes. Pozrel na obrazovku a zistil, že mu chýbajú už len tri. Na chvíľku zaváhal, či si nedá pol hodinku relaxu pomocou mozgového elektrovitatransfokátora, ale potom si povedal, že to radšej odloží na večer. Prezrel si ďalšie eseje a zistil, že nasledujúce dve sú o rovnakej téme. Začítal sa teda do oboch.

Diagnóza: spolupráca?

Kamil Krnáč
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
kamil.krnac@kemco.sk

Abstrakt. Softvéroví inžinieri by mali byť čestní a nápomocní k ich spolupracovníkom (kódex softvérového inžiniera organizácií ACM a IEEE, verzia 5.2). Pre prácu v pracovných spoločenstvách sú pravidlá spolupráce z morálneho aspektu aplikované zo všeobecných poznatkov o medziľudských vzťahoch, na základe ktorých sa definujú etické princípy. Je preto potrebné poukázať na prínos a obmedzenia vyplývajúce zo spoločenských „mantinelov" definovaných napr. aj etickými pravidlami softvérového inžiniera

Etický kódex softvérového inžinierstva ACM a IEEE, 7. princíp: Spolupracovníci

Softvéroví inžinieri by mali byť čestní a nápomocní k ich spolupracovníkom [1]. Obzvlášť by, podľa nasledovného, mali:

7.01. Podnecovať spolupracovníkov k dodržiavaniu týchto odporúčaní.
7.02. Podporovať spolupracovníkov v profesionálnom rozvoji.
7.03. Plne oceniť zásluhy iných a vyhýbať sa nenáležitému prospechu.
7.04. Kritizovať prácu iných nestranne, spravodlivo a riadne dokumentovaným spôsobom.
7.05. Korektne vypočuť názory, znepokojenia alebo sťažnosti spolupracovníka.
7.06. Napomáhať spolupracovníkom dostať do povedomia súčasné štandardné predpisy včítane stratégií a procedúr ochraňujúcich heslá, súbory a iné dôverné informácie a bezpečnostné opatrenia vo všeobecnosti.
7.07. Nezasahovať nečestne do kariéry spolupracovníkov, avšak, v dobrej viere, v záujme zamestnávateľa, zákazníka alebo verejného prospechu, preveriť spôsobilosť spolupracovníka.
7.08. V situáciách mimo vlastných oblastí kompetencií požiadať o názory profesionálov v týchto oblastiach kompetentných.

Úvod

Dlhodobými pozorovaniami zo sveta softvérových projektov, ale takisto aj z iných oblastí sa dospelo k uzáveru, že najefektívnejšia forma dosahovania vytýčených cieľov je založená na spolupráci. Podstatou spolupráce je synergický efekt, pomocou ktorého sa dosahuje väčší objem hodnôt pre jednotlivca ako pri individualistickom, súťaživom princípe. Organizácia závisí od podmienok a prostredia, vo všeobecnosti je však možné pozorovať skupinovú prácu, tímovú prácu a pod. Všeobecne teda môžeme hovoriť napríklad o *pracovnom kolektíve*. Pre prácu v takýchto spoločenstvách sú pravidlá kooperácie z morálneho aspektu aplikované zo všeobecných poznatkov o medziľudských vzťahoch, na základe ktorých sa definujú etické princípy poukazujúce na eticky správny postoj k spolupráci v pracovných procesoch. Pozrime sa teda bližšie na časť týchto zásad etického správania sa softvérového inžiniera.

Čo skrývajú etické princípy vo vzťahu k spolupracovníkom?

Súčasný svet je svetom rýchlych, dynamických zmien, kde každé zaváhanie môže na jednej strane znamenať dosiahnutie cieľa bezpečnejšou a istejšou cestou, na druhej strane však môže spôsobiť zaostanie za vývojom a premeškanie príležitostí. Tento trend sa prejavuje aj v správaní sa ľudí v bežnom pracovnom procese a z tohto pohľadu je otázka napr. podpory profesionálneho rozvoja u spolupracovníkov postavená pod drobnohľad na jednej strane etiky vo vzťahu ku globálnemu profesionálnemu vzrastu (napr. celej firmy) a na druhej obáv individualít z predčasnej nahraditeľnosti schopnejšími a asertívnejšími jedincami. Z pohľadu celku býva v otázkach profesionality prístupu k pracovnému procesu uprednostňovaný vzrast globálny pred vzrastom založeným čisto individualisticky, a teda z tohto uhla je dodržiavanie etických princípov dôležité a nespochybniteľné. V praxi sa však vyskytuje veľmi častý jav odchyľovania sa od týchto etických zásad v spojení s prejavmi konkurenčného úsilia vyniknúť a byť úspešnejší, čo vedie ku krokom niekedy buď hraničiacim s porušovaním etiky alebo k úplným rozporom s morálnym kódexom. Je preto potrebné zamyslieť sa nad touto problematikou a poukázať na prínos a obmedzenia vyplývajúce zo spoločenských „mantinelov“ definovaných napr. aj etickými pravidlami softvérového inžiniera.

Podpora spolupracovníkov v profesionálnom rozvoji

Spôsobov na podporu profesionálneho rozvoja je mnoho a zväčša sa jedná o podporu zhora nadol, t.j. napr. nadriadení pracovníci zabezpečujú školenia, kurzy a tréningy pre svojich podriadených, pričom veľmi dôležitú úlohu hrá podpora sebarealizácie. Ťažko si predstaviť dlhodobejšiu užitočnosť kvalitných profesionálnych pracovníkov, pokiaľ by sa ich schopnosti a nadanie neprehlbovali

alebo pokiaľ by člen pracovnej skupiny (tímu) prejavil potrebu väčšieho osobnostného priestoru pre svoj rast a tento prejav by bol chtiac alebo nechtiac utlmovaný [2]. Vplyv nadriadených pracovníkov na podriadených je z hľadiska rozvoja nutný a jeho kvalita je potom zúročená výsledkami, efektivitou práce a entuziazmom pracovníkov. Pomerne častá je takisto podpora spolupracovníkov na tej istej úrovni v hierarchii pracovných vzťahov, ale väčšinou sa jedná napríklad o prvotné zaškolenie nového pracovníka do systému, resp. pomoc pri zvládnutí základných problémov (napr. pomoc v používaní programovacieho jazyka v tej konkrétnej oblasti). Nazvime nového pracovníka *kandidátom* [3]. Pomerne silná väzba kandidáta voči pracovnému prostrediu sa dosahuje pridelením kandidáta určitému členovi pracovného kolektívu, s ktorým vytvorí tandem na riešenie nejakej podúlohy. Výhodou je, že dochádza ku kopírovaniu hodnotných vlastností člena kolektívu na kandidáta, preto je veľmi dôležitý výber osoby, ku ktorej bude kandidát pridelený. V reálnych pracovných kolektívoch, kde sú už pomery ustálené a zabehnuté, teda sú definované pozície, funkcie, úlohy je naopak zriedkavým javom priama podpora rozvoja medzi spolupracovníkmi práve z už uvedených dôvodov existujúcej potenciálnej profesionálnej konkurencie. Zdravá forma súťaživosti je samozrejme vítaná, pretože sama o sebe podporuje motivačný prístup k rozvoju profesionálnych vlastností, ale egoistický a uzavretý prístup má svoje hranice, pokiaľ nezačne vplývať demotivujúcimi prvkami na prácu v kolektíve. Dôležitý faktor úspechu (softvérového) projektu teda spočíva v *podpore spolupracovníkov v profesionálnom rozvoji*. Ďalším z faktorov, ktoré vplývajú všeobecne na rozvoj osobnosti je ocenenie úspechov.

Ocenenie zásluh iných a vyhýbanie sa nenáležitému prospechu

Vo svojej knihe *Neznamenám mnoho, ale som taký, ako som bol stvorený* hovorí psychológ Jess Lair: “Chvála je ako slnečný lúč, ktorý zahreje ľudskú dušu a bez nej sa nemôžeme rozvíjať a rozkvitať”.

Pozitívne hodnotenie je motorom motivácie, ktorá je prevenciou proti odkladaniu, „odfláknutiu“ práce, problémom s dochádzkou a produktivitou alebo zmeškaniu termínov [4]. Ocenenie zásluh je napriek tomu veľmi citlivou záležitosťou, pretože je nutné zvážiť, v akej forme sa ocenenie prezentuje. V extrémnych prípadoch sa môže stať, že ocenenie má presne opačný účinok a vyznieva negatívne. Je to v prípadoch, keď sa napríklad za zásluhy kolektívu ocení jednotlivec, alebo ak je ocenenie vypočítavé, alebo ak podnet naň pramení skôr z osobných dôvodov ako z profesionálneho ohodnotenia. Preto by sa prípadné verejné ocenenie malo vždy spájať s úspechom celého pracovného kolektívu [2] a osobné individuálne ocenenia by zväčša mali byť predmetom uzavretých stretnutí. Dôležité je vyslovovať konkrétne uznania namiesto všeobecnej pochvaly. Pre dotyčného znamenajú vyzdvihnuté konkrétne úspechy omnoho viac a bude ocenenie považovať skôr za úprimné, ako keby bolo nahradené všeobecnými frázami o všeobecných výsledkoch a faktoch [5]. Obzvlášť demotivujúcim prvkom, ktorý

obmedzuje priestor profesionálneho rozvoja je privlastňovanie si cudzích úspechov. Na zvyšok kolektívu (tímu, pracovnej skupiny...) pôsobí tento jav veľmi negatívne a môže viesť k úplnému zastaveniu pracovného alebo duševného vývoja kolektívu resp. k jeho úplnej deštrukcii. Tieto javy sú viditeľné aj v softvérových projektoch, kde je teda dôležité, aby softvéroví inžinieri *plne ocenili zásluhy iných a vyhýbali sa nenáležitému prospechu.* Ruka v ruke s korektným oceňovaním a pochvalami by však mala kráčať konštruktívna nestranná kritika.

Kritika práce iných nestranná, spravodlivá a riadne dokumentovaná

Správne kritizovať nie je jednoduché. Ľudia sú od svojej podstaty veľmi citliví na kritiku, pretože ju považujú za útok resp. poškvrnenie hrdosti. Je teda potrebné nájsť správny prístup ku poukazovaniu chýb alebo nedostatkov iných. Zadefinujme si pojem *zdravého pracovného kolektívu*. V takomto kolektíve je kritika bežným pracovným nástrojom pri tvorivej práci za predpokladu, že sú dané podmienky, za akých sa kritika prezentuje. Členovia kolektívu sú na kritiku zvyknutí a nemajú pocit, že sa jedná o znehodnotenie ich úsilia, pričom mnohokrát si túto kritiku priamo vyžadujú. Atribútmi správnej kritiky sú: konštruktívne námety podložené reálnymi argumentmi a prípadné návrhy na riešenia alebo vylepšenia. Je vhodné, keď je diskusia o probléme položená medzi viacerých členov tímu, pretože "súboj jeden na jedného" môže ľahko skĺznuť do roviny individuálnych sporov, ktoré nemusia viesť ku konsenzuálnemu uzáveru. Ak má možnosť vyjadrenia väčší počet účastníkov, väčšinový názor (po zvážení všetkých dostupných argumentov) je zväčša ľahšie prijateľný ako názor jedinej osoby, ktorá sa môže situovať v podvedomí kritizovanej osoby ako útočník. To vedie k neopodstatneným a obvykle neadekvátnym obranným reakciám, ktoré môžu viesť k zdeformovanému pohľadu na situáciu.

U pracovných kolektívov, ktoré nemožno nazvať „*zdravými*" je častým problémom negácia akýchkoľvek aj konštruktívnych podnetov poukazujúcich na nedostatky alebo chyby spolupracovníkov a mohutnosť obranných reakcií kritizovaného je niekedy priamoúmerná času a úsiliu, ktoré danej úlohe venoval. Je to zvyčajne prípad ľudí, spolupracujúcich dlhšiu dobu, medzi ktorými vznikli negatívne osobné vzťahy prenášajúce sa aj do pracovných problémov, čo vytvára podmienky pre zaujatú a osobne ladenú kritiku resp. jej prijímanie. Čo je teda dôležité pre správnu kritiku? Na chyby treba upozorňovať nepriamo [5], nerozpitvávať zbytočne predmet kritiky a hlavne dať kritizovanému priestor na reakciu (viď. 3.4). Pokiaľ je predchodcom negatívnych postojov a podnetov k vylepšeniu práce spolupracovníka poukázanie na pozitívne skutočnosti, pochvala či uznanie za konkrétne výsledky, alebo dokonca súčasná zmienka o vlastných chybách a problémoch, sú tieto dodatočné negatívnejšie hodnotenia prijímané oveľa jednoduchšie, čo dosť výrazne zabraňuje vzniku prehnaných obranných bariér kritizovanej osoby. Proces korektnej spätnej väzby medzi spolupracovníkmi je ukrytý v *nestrannej, spravodlivej a riadne dokumentovanej kritike* a jej rovnako

objektívnom prijímaní. Nedeliteľnou súčasťou správnej kritiky je teda aj zistenie pohľadov a názorov kritizovanej strany.

Korektné vypočutie názorov, znepokojení alebo sťažností spolupracovníka

Známe úslovie hovorí, že dva monológy ešte nie sú dialógom. Správna komunikácia pozostáva nielen z vysielania, ale aj z prijímania informácii, pričom schopnosti dobre zvládať proces prijímania informácii sa nazýva aj „umením načúvania" [6]. V bežnom pracovnom prostredí sa vyskytujú problémy, ktoré nemusia byť na prvý pohľad viditeľné. Je prínosom, ak sa akýkoľvek podnet vychádzajúci od ľubovoľného člena pracovného kolektívu nenechá nepovšimnutý, pretože sa môže jednať o podstatnú informáciu niekedy spôsobujúcu radikálne zmeny v smerovaní ďalších krokov pracovného procesu. Môže ísť o problémy odborného charakteru, kde je potrebné zvážiť odlišné názory spolupracovníkov alebo osobné problémy spôsobujúce zhoršené pracovné prostredie pre dosiahnutie vytýčených cieľov. Problémom globálnejšieho charakteru sú venované porady a naopak individuálne stretnutia riešia problémy užšieho rozsahu, prípadne háklivé témy osobného rázu.

Dôležité parametre korektného vypočutia názorov spolupracovníkov sú napr.: prejavenie záujmu a vžitie sa do problémov, udržanie rešpektu a dôstojnosti počúvaného [6]. Počúvaný by mal dostať dostatočný priestor na prezentáciu svojich názorov a návrhov, pričom aplikácia prípadných protiargumentov počúvajúceho formou otázok je vhodnejšia ako nastolenie priamych negatívnych uzáverov alebo direktívnych opravných tvrdení. Bez korektnej komunikácie sa nemôžu dostaviť želané výsledky a dosiahnutie cieľov je brzdené existenciou "tichých problémov" alebo osobnostných konfliktov brániacich konštruktívnym riešeniam situácií. V zdravom pracovnom kolektíve je teda z hľadiska profesionálneho vývoja a rovnováhy vzťahov alfou a omegou spätná väzba formou *korektného vypočutia názorov, znepokojení alebo sťažností spolupracovníkov*.

Záver

Medziľudské vzťahy sú rozmanité a komplikované. V pracovných procesoch sa táto rozmanitosť prejavuje rôznymi pozitívnymi i negatívnymi väzbami medzi spolupracovníkmi a cieľ spočívajúci v zavedení pravidiel, akými sú napríklad body etického kódexu softvérového inžiniera, je v prínose morálnych zásad a odporúčaní do vzájomných vzťahov medzi spolupracovníkmi. Ich význam zasahuje do bežného správania sa v pracovnom kolektíve svojimi ohraničeniami, pretože striktne dodržiavať tieto zásady je niekedy obtiažne vzhľadom na situáciu alebo prostredie. Napriek tomu sú však dané pravidlá akousi definíciou kvalitných medziľudských vzťahov a v pracovných kolektívoch softvérových inžinierov na profesionálnej úrovni by mali byť každodenne používaným nástrojom.

Použitá literatúra

[1] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[2] Jim McCarthy: Softwarové projekty. Computer Press, 1999. (ISBN 80-7226-164-0)

[3] Manažér, časopis pre rozvoj riadiacich pracovníkov 3/98, ročník tretí, 1998.

[4] Mária Bieliková: Manažment v softvérovom inžinierstve. Slovenská technická univerzita, 1999.

[5] D. Carnagie: Jak získávat přátele a působit na lidi. TALPRESS, Praha 1993. (ISBN 80-85609-12-6)

[6] Jiří Plamínek: Řešení konfliktů a umění rozhodovat. Argo, 1994. (ISBN 80-85794-14-4)

Leto s bohmi

Viktor Zeman
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
zemanv@decef.elf.stuba.sk

Abstrakt. Esejou by som rád reagoval na časť definície etického kódexu softvérového inžiniera organizácií ACM a IEEE Computer Society [2] s názvom Spolupracovníci (verzia 5.2). Mojím cieľom a úlohou je pochopiť a zároveň pokúsiť sa vysvetliť etické odporúčania vypracované v spomenutom kódexe. Hlavnou myšlienkou časti s názvom Spolupracovníci je odporúčanie pre softvérového inžiniera byť čestný a nápomocný ku spolupracovníkom. Na tieto princípy sa budem snažiť poukázať v mojom skutočnom príbehu z prázdnin, ktorý dokazuje, že dodržiavanie etického kódexu nemusí byť len ďalším obmedzením v živote softvérového inžiniera.

Etický kódex softvérového inžinierstva ACM a IEEE, 7. princíp: Spolupracovníci [2]

Softvéroví inžinieri by mali byť čestní a nápomocní k svojim spolupracovníkom. Obzvlášť by si mali osvojiť nasledovné pravidlá:

7.01. Podnecovať spolupracovníkov k dodržiavaniu týchto odporúčaní.
7.02. Podporovať spolupracovníkov v profesionálnom rozvoji.
7.03. Plne oceniť zásluhy iných a vyhýbať sa nenáležitému prospechu.
7.04. Kritizovať prácu iných nestranne, spravodlivo a riadne dokumentovaným spôsobom.
7.05. Čestne vypočuť názory, znepokojenia alebo sťažnosti spolupracovníka.
7.06. Napomáhať spolupracovníkom, byť si plne vedomý súčasných štandardných
predpisov včítane stratégií a procedúr ochraňujúcich heslá, súbory a iné dôverné informácie a bezpečnostných opatrení vo všeobecnosti.
7.07. Nie nečestne zasahovať do kariéry niektorého zo spolupracovníkov, avšak ak si to
vynútia záujmy zamestnávateľa, zákazníka alebo verejného prospechu, v dobrej viere preveriť spôsobilosť spolupracovníka.

7.08. V situáciách mimo vlastných oblastí kompetencií požiadať o názory profesionálov,
ktorí majú kompetencie v týchto oblastiach.

Ako sa to začalo?

Mal som šťastie. Všetko sa to začalo vyplnením žiadosti, do ktorej som veľa nádejí nevkladal. Letná prax vo Viedni bola pre mňa atraktívnou príležitosťou skúsiť niečo „lepšie". Nádej ma opúšťala už na začiatku z toho dôvodu, že som nevedel ani len „ceknúť" po nemecky. Čo ale náhoda nechcela, dostal som sa do oddelenia SIEMENS PSE KBA. Už názov vo mne vzbudzoval rešpekt. Keďže som už aj predtým pracoval na väčších projektoch na Slovensku, mal som skúsenosti so vzťahmi na pracovisku a aj mimo neho. Boli viac-menej kladné, ale vyskytli sa aj situácie, kedy by som vedel povedať, že nie všetko bolo tak, ako by to malo byť.

Viac ľudí ma strašilo, že v zahraničí sa na pracovisku nepestujú veľmi kamarátske vzťahy a cudzinec sa len veľmi ťažko vkliní do zohraného kolektívu. Uisťovali ma o tom dokonca aj pracovníci, ktorí už pár rôčkov v zahraničí pracovali. Skončilo to tým, že som tomu takmer uveril a v deň nástupu do práce som bol dosť nervózny. Moju nervozitu podnecoval aj fakt, že sa odo mňa očakávali znalosti, ktoré som nemal. Bol som pridelený na projekt vytváraný v systéme Lotus Notes, o ktorom som vtedy počul prvý raz.

Asi takto sa začali tri mesiace, ktoré som strávil za Dunajom....

Uplatňovanie etického kódexu

Keď si to z dnešného pohľadu uvedomím, stal som sa "dokonalým terčom" uplatňovania etického kódexu (samozrejme v dobrom zmysle). Už v prvý deň si na mne veľká časť mojich budúcich spolupracovníkov vyskúšala pár bodov kódexu, takže som sa na mojom novom pracovnom mieste dokonale udomácnil. Keďže s nemčinou som vyzeral ako nemluvňa, s pochopením celého pracovného tímu sme prepli jazyky (a poniektorí aj slovníky) na angličtinu.

Vývoj udalostí počas letných prázdnin sa takmer zázračne zhoduje s bodmi v opisovanej časti etického kódexu. Preto sa pokúsim môj prázdninový príbeh rozdeliť do podobných častí.

Podnecovať spolupracovníkov k dodržiavaniu týchto odporúčaní. Myslím si, že spolupracovníci v tíme, do ktorého som sa dostal, etický kódex nečítali. Avšak nálada v tíme podnecovala každého z nás konať tak, aby si tím udržal aj naďalej dobré meno. Pravidlo číslo 1 sa tým akosi samovoľne napĺňalo. Až teraz si uvedomujem, že týchto pár pravidiel, ktoré sme nevedomky dodržiavali, nám dovolilo pracovať vo vysokom nasadení, čo sa prejavilo aj na spokojnosti zákazníkov.

Podporovať spolupracovníkov v profesionálnom rozvoji. Prvý mesiac bol pre mňa prelomový. Musel som sa naučiť od základov všetko o systéme, v ktorom bol projekt vytváraný. Druhé pravidlo si zobrali ostatní členovia za svoje a venovali mi toľko času a vedomostí, o koľko som ich požiadal. Musím podotknúť, že toho nebolo málo. Dokonca si myslím, že chvíľami som im svojimi otázkami riadne znepríjemňoval život. Ale nakoniec sa im to oplatilo. Na moje veľké potešenie som po mesiaci tvrdej driny zistil, že už sa zo mňa stal celkom dobrý odborník v danej oblasti. Na prvý pohľad to vyzerá jednoducho, ale z mojej strany si to vyžadovalo poriadnu dávku sebazapierania. Denne som sedával pri počítači aj viac ako 14 hodín. Tým som chcel ukázať tímu, že ich snaha nie je márna a záujem na zdarnom výsledku mám aj ja.

Plne oceniť zásluhy iných a vyhýbať sa nenáležitému prospechu. Po prvom mesiaci sa moje znalosti museli podrobiť ťažkej skúške. Dostal som na starosť nový projekt, ktorý bol rozsahom menší, ale o to dôležitejší pre vedenie spoločnosti. Projekt bol iniciovaný v čase dovoleniek, keď trvalí zamestnanci ležali na piesočnom pobreží opaľujúc sa na žiarivom slnku. Čo čert nechcel, zrazu som bol jediný, ktorý to bol schopný spraviť a mal čas. A tak sa začal opäť kolotoč presedených dní, opaľujúc sa len pred rozžiareným monitorom. Koniec projektu bol však famózny. (Samozrejme, že som ho nedokončil len vďaka tomu, že som dlho nad ním sedel. Veľký podiel na výsledku mali opäť tí, čo mi neprestajne radili.) Po prvýkrát ma pochválil niekto z druhej línie celej firmy. Bolo to pre mňa až neuveriteľné, že človek, o ktorom som sa normálne mohol dočítať len z vnútropodnikových časopisov si našiel čas na nevýznamného študenta a stratil s ním pár pochvalných slov. Práve takáto zdanlivo nevýznamná pochvala znamená pre človeka veľakrát oveľa viac ako riadne zvýšenie platu. Dodržiavaním tretieho pravidla v kolektíve pracovníkov vzniká podľa môjho názoru prirodzený motivačný ťah, ktorý posúva projekty míľovými krokmi vpred.

Kritizovať prácu iných nestranne, spravodlivo a riadne dokumentovaným spôsobom. Ako v každom programe, tak aj v mojom sa nakoniec vyskytli chybičky. Úprimne obdivujem, akým spôsobom na chyby v mojom programe poukázali spolupracovníci. Nesnažili sa mi dokázať, že moja chyba ukazuje na to, že nemám potrebné kvality. Práve naopak, snažili sa ma motivovať k ešte väčšiemu výkonu. Všetko čo sme spolu vyvíjali, sme prebrali na spoločných stretnutiach, ktoré sa konali vždy, keď si to situácia vyžadovala. Na stretnutiach sa zúčastňovali aj tí, ktorí priamo s daným problémom nemali nič spoločné. Kolektívne stretnutia mali jednu veľkú výhodu. Človek nebol postavený len pred kritiku jednotlivca, ale pred kritiku celej skupiny, čím sa zmenšovala šanca, že si kritiku bude brať osobne. A čo bolo hlavné, za kritikou vždy nasledovali konštruktívne nápady, ktorými sme sa snažili všetci spolu riešiť problém. Výsledkom takýchto stretnutí (niekedy len 10-minútových) bolo to, že kritizovaný človek sa necítil ako najväčší chudák, ale ako člen tímu, ktorý vyriešil problém. Tento jav som nezažil zatiaľ ani na jednom z projektov, na ktorých som sa doposiaľ podieľal. Myslím si, že takto by mal postupovať každý tím, ktorý chce svoju prácu dokončiť úspešne.

Čestne vypočuť názory, znepokojenia alebo sťažnosti spolupracovníka. Úspešné riešenie problémov však nezáležalo len na tom, ako sa podala kritika. Aby mala

kritika nejaký zmysel, musí padnúť na úrodnú pôdu u pozorného poslucháča, ktorý si buď uvedomí svoju chybu, alebo dokáže rozumne oponovať. Podobne túto myšlienku sformoval aj Dale Carnagie [1], ktorý tvrdí: „Ak sa zmýlite, uznajte svoj omyl rýchlo a ochotne." Podobne to platilo aj v našom mladom kolektíve. Ak sme na spoločných stretnutiach zistili, že sa niekto z nás zmýlil, nesnažili sme sa za každú cenu obhajovať zlé riešenia len preto, aby si niekto zachoval čistý štít.

Napomáhať spolupracovníkom, byť si plne vedomý súčasných štandardných predpisov včítane stratégií a procedúr ochraňujúcich heslá, súbory a iné dôverné informácie a bezpečnostných opatrení vo všeobecnosti. Informácie, informácie, informácie. To bol artikel, s ktorým som mohol narábať a ktorý som mohol zneužiť. Samozrejme, že som musel sľúbiť, že ich v žiadnom prípade nezneužijem. Prízvukovali mi to cez leto niekoľkokrát každý týždeň. Naozaj kreatívny prístup k ochrane informácií však predviedol náš projektový vedúci. Keďže bolo potrebné modifikovať databázu obsahujúcu dôverné informácie o rozpočte jednotlivých oddelení spoločnosti a zároveň nechcel, aby sa tieto informácie dostali medzi ľudí, dal modifikovať túto databázu práve mne. Nič by na tom nebolo zaujímavé, ak by som vedel trošku nemecky. Databáza bola vytvorená v nemeckom jazyku, takže aj keď som pri modifikácii videl všetky informácie, nevedel som si ich správne vysvetliť. Aj keď tento príklad priamo nepoukazuje na bod z etického kódexu, myslím si, že je ukážkou toho, ako sa dá k ochrane dôverných informácií pristupovať aj netradičným spôsobom.

Nie nečestne zasahovať do kariéry niektorého zo spolupracovníkov, avšak ak si to vynútia záujmy zamestnávateľa, zákazníka alebo verejného prospechu, v dobrej viere preveriť spôsobilosť spolupracovníka. Pri riešení projektov bol vždy veľký problém s ľudskými zdrojmi. Projektov bolo vždy viac ako pracovníkov, takže každý pracoval súčasne na viacerých projektoch. Aj z tohto dôvodu sa organizácia tímu nedelila striktne na šéfa a podriadených, ale za každý projekt v našom tíme zodpovedal iný človek. Malo to viac výhod. Každý projekt bol udržiavaný jedným človekom, ktorý ak potreboval pomoc, zaangažoval ostatných spolupracovníkov, koordinoval ich činnosť a zodpovedal za výsledky a termíny v projekte. Pri takejto organizácii práce bola menšia šanca, že sa v niektorom projekte na niečo pozabudne a projekt bude neúspešný. Distribuované riadenie malo aj tú výhodu, že dovoľovalo jednotlivcom sa presadzovať a budovať kariéru. Ak bol niektorý z projektov úspešný, tak to znamenalo, že človek, ktorý ho vedie, je schopný. Asi takto by som si predstavoval posilňovanie autority členov tímu, o ktorom píše pán McCarthy vo svojej knihe [3].

Napriek mnohým výhodám mal aj tento model riadenia malé nedostatky. Problém nastal, ak projekt dostal na starosť človek menej priebojný a možno chvíľami až lenivý. Prejavilo sa to ihneď problémami s termínmi plnenia projektového plánu. Nášmu tímu sa stala takáto situácia s projektom, ktorý mal na starosť človek bez vzťahu k práci. Tím podvedome cítil, že niečo nie je v poriadku a dávali sme to patrične najavo nášmu „lenivejšiemu" kolegovi. Po mesiaci problémov to v tíme už takmer vrelo kvôli problémom, ktoré spôsoboval sústavne jediný človek. Rozhodnutie o osude nášho spolupracovníka sme vyriekli aj pred hlavným šéfom, ktorý uznal, že naše požiadavky na odvolanie sú opodstatnené. Pôvodne sme ako tím

žiadali, aby tento pracovník bol prepustený z našej firmy. Náš vedúci však situáciu vyriešil naozaj bravúrne. Pracovníkovi, ktorému sa "nedarilo" v našom tíme, ponúkol možnosť vyskúšať si svoje schopnosti v inom tíme na inom oddelení a diametrálne inom projekte. Bolo to naozaj férové riešenie, ktoré zodpovedá tomuto bodu z etického kódexu. Aj keď kariéra tohto pracovníka v našom tíme nebola nijako posilnená, ba až upadala, dostal druhú šancu, ktorú, dúfajme, využije lepšie.

V situáciách mimo vlastných oblastí kompetencií požiadať o názory profesionálov majúcich kompetencie v týchto oblastiach. Náš tím sa venoval aj problémom v komunikácii medzi sociálne postihnutými ľuďmi. Vyvíjali sme systém, ktorý by im uľahčil práve túto komunikáciu. Systém sme museli navrhnúť tak, aby postihnutí ľudia s ním vedeli narábať a hlavne aby ho vedeli pochopiť. Po zahájení projektu sme začali navrhovať jednotlivé časti systému. Podarilo sa nám dokončiť funkčný prototyp, ktorý sme nasadili do skúšobnej prevádzky, aby sme otestovali jeho kvality. Po prvých dňoch prevádzky sme však zistili, že psychologický aspekt ľudí, ktorí mali používať náš systém, vôbec nebol zapracovaný do projektu. Náš prvý prototyp sa stretol s odmietnutím používateľov pre jeho až prílišnú „dokonalosť". Zistili sme, že ľudia, ktorým mal byť určený, nie sú schopní porozumieť takémuto komplexnému systému, aj keď nám sa zdal jednoduchý. Jediným možným východiskom v danej situácii bolo nadviazanie spolupráce so sociálnymi pracovníkmi, ktorí pracovali s ľuďmi, rozumeli ich potrebám a vedeli sa vžiť do ich životnej situácie. Spolupráca s ľuďmi, ktorí boli v tejto oblasti „doma" sa osvedčila a projekt bol nakoniec úspešne zavedený do domácností postihnutých ľudí. Aj keď ako tím softvérových inžinierov sme mali vedomosti o technikách na vytvorenie systému, bez pomoci odborníkov by takýto multidisciplinárny projekt určite nebol dokončený. Z tohto útržku mojich zážitkov vyplýva, že každý človek by mal robiť to, v čom je dobrý, a o problémoch z iných oblastí života by sa mal radiť s kvalifikovanejšími.

Všetko riešte priateľsky. Podľa môjho názoru bod o priateľstve v pôvodnej definícii etického kódexu chýba. Myslím si, že práve priateľské vzťahy v našom tíme nám dovolili vytvoriť niekoľko úspešných projektov. Toto pravidlo by malo byť určené nielen pre softvérových inžinierov, ale malo by platiť aj v každodennom styku s ľuďmi.

Záver

Mojím príbehom som chcel poukázať na to, že dodržiavanie etického kódexu nemusí byť vždy len jedným z ďalších nepríjemných obmedzení a pravidiel, s ktorými sa v živote stretávame. Spomedzi mnohých článkov publikovaných na internete vysvetľujúcich modelové situácie použitia zásad etického kódexu, som našiel len málo takých, ktoré by vysvetľovali etický kódex na kladných príkladoch. Zväčša sa autori snažia poukazovať na negatívne príklady, ktorým by sme sa mali vyhnúť. Keďže som chcel byť v tomto smere trošku iný, pokúsil som sa vysvetliť časť etického kódexu venovanú spolupracovníkom na kladnom príklade mojich skúseností.

Pri premýšľaní o obsahu mojej reakcie na časť etického kódexu som si uvedomil, že nemá zmysel umelo vytvárať vymyslené príbehy, keď som podobný sám prežil. Prajem každému softvérovému inžinierovi, aby mal možnosť pracovať v rovnako „božskom“ kolektíve, ako som mal možnosť pracovať ja.

Použitá literatúra

[1] Carnagie D.: Jak získávat přátele a púsobit na lidi. TALPRESS, Praha 1993. (ISBN 80-85609-12-6)

[2] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[3] Jim McCarthy: Softwarové projekty. Computer Press, 1999. (ISBN 80-7226-164-0)

Wesly sa po prečítaní posledných dvoch esejí pousmial. Do centrálnej databázy recenzií článkov napísal len krátku poznámku. „Článok sa venuje problémom, ktoré súvisia so spoluprácou v pracovných kolektívoch na začiatku storočia. Autor opisuje hlavne etické aspekty spolupráce jednotlivcov. Etické problémy pramenia z povahy vtedajšieho pracovného kolektívu.“ Nevedel si predstaviť, ako to mohlo vyzerať v pracovnom kolektíve, ktorý opisovali autori esejí. Wesly bol veľmi precízny pri štúdiu histórie a tak mu nedalo aby si neprečítal o vývine pracovného kolektívu viac. Pohodlnejšie sa usadil v kresle a začal sa prehrabávať v GDIS (global data inforamtion storage). V historickej videobanke objavil multimediálny dokument z roku 2034 od Dr. Adamskeho, ktorý niesol názov „Vývin softvérového pracovného kolektívu a spôsob práce v minulom storočí“. Hoci nemal mnoho času, rozhodol sa, že si ho prezrie.

Dokument obsahoval precíznu štúdiu o vývine spôsobu práce a kolektívu v softvérových firmách. Wesly ho rýchle prebehol a narazil na kľúčový bod, ktorým bol rok 2030, kedy prof. Minsky dokázal exaktne definovať psychologické modely správania sa jednotlivcov čo sa dovtedy pokladalo za nemožné. Nemožné to bolo najmä z dôvodu obrovskej zložitosti a potreby gigantickej simulácie. Tento problém odstránili nové počítače CRAYON 7997 založené na supra-antivodivostnom princípe. Minskeho práca a niekoľko ďalších objavov viedli k tomu, že pracovné kolektívy sa už nevytvárajú chaoticky, ale ich vytváraniu predchádza počítačová simulácia správania sa a interakcie jednotlivcov. Takáto štúdia, ktorá je dnes pri prijímaní jednotlivca do tímu úplne bežná, má za cieľ odhaliť, či je jednotlivec vhodný pre daný kolektív a aké sú potenciálne riziká jeho prijatia. Navyše svetová globalizácia viedla k tomu, že klasický softvérový tím, tak ako ho poznali v minulom storočí sa pretransformoval na virtuálnu komunitu spolupracujúcich ľudí, ktorí sa v reálnom svete vôbec nemusia stretnúť. Technológie hmotných holografických projekcií viedli k tomu že ľudia dokážu dokonale kooperovať aj bez potreby vzájomne sa stretnúť.

Softvérové tímy sú dnes vedené špeciálne zaškolenými jednotlivcami, čo vyžaduje zákon o softvéri, aj keď v poslednom čase sa začalo šuškať, že za niektorými tímami stoja umelo vytvorení softvérový agenti.

Záhradkár

Robert Kovaľ
Slovenská technická univerzita, Fakulta elektrotechniky a informatiky
Ilkovičova 3, 831 02 Bratislava, Slovenská republika
rkoval@gratex.sk

Abstrakt. V súčasnom období vystupuje do popredia otázka etiky softvérového inžiniera. Už v minulosti bolo vytvorených niekoľko etických kódexov, ktoré sa snažili definovať princípy etiky súvisiace špecificky s oblasťou softvérového inžinierstva. V tejto eseji sa snažím reagovať na kódex, ktorý bol vyvinutý po vzájomnej spolupráci organizácií ACM a IEEE-CS, Software engineering code of ethics and professional practice (verzia 5.2). Špeciálne sa esej venuje ôsmemu bodu tohto kódexu – vzťahu softvérového inžiniera ku sebe samému a snaží sa poukázať na potrebu zavedenia etických princípov do praxe.

Etický kódex softvérového inžinierstva ACM a IEEE, 8. princíp: Vzťah k sebe

Softvéroví inžinieri by sa mali celoživotne vzdelávať pri vykonávaní svojej profesie a mali by dodržovať etický prístup. Podrobnejšie povedané, softvérový inžinier by sa mal neustále snažiť [1]:

8.01. O rozširovanie svojich vedomosti o vývojových tendenciách analýzy, špecifikácie, návrhu, údržby, testovania softvéru a s nimi spojenej dokumentácie, spolu s manažmentom procesu ich vývoja.
8.02. Vylepšovať svoje schopnosti vyvíjať bezpečný, spoľahlivý, použiteľný a kvalitný softvér pri rozumných nákladoch a v rozumnom čase.
8.03. Vylepšovať svoje schopnosti pri vytváraní presnej, obsažnej a kvalitnej dokumentácie.
8.04. Vylepšovať svoje znalosti o softvéri, na ktorom pracuje a s ním súvisiacich dokumentoch a tiež o prostredí, v ktorom bude softvér použitý.
8.05. Vylepšovať svoje vedomosti o štandardoch a zákonoch súvisiacich s vyvíjaným softvérom a príslušnými dokumentmi.
8.06. Vylepšovať svoje vedomosti o etickom kódexe, jeho interpretácii a uplatňovaní pri jeho práci.
8.07. Nezaobchádzať s nikým nespravodlivo z dôvodu svojich predsudkov voči danej osobe.
8.08. Nenavádzať nikoho na porušovanie tohto kódexu.

8.09. Uvedomiť si, že porušovanie tohto kódexu sa nezlučuje s bytím profesionálnym softvérovým inžinierom.

Úvod

Problematika etiky softvérového inžiniera vo vzťahu k sebe je veľmi problematickou a to hlavne pre rozpor medzi záujmom profesionálnym a záujmom osobným. Tento rozpor často spôsobuje porušovanie etického kódexu a úzko súvisí s psychológiou jednotlivca. Jednotlivec – softvérový inžinier by sa aj napriek tomuto rozporu mal snažiť kódex dodržiavať a viesť k tomu aj svoje okolie. V tejto súvislosti sa vynára ešte ďalšia otázka, ktorá podľa mňa s etikou úzko súvisí a to je kvalita života osoby. Nech už je definícia kvality života akákoľvek, porušovanie etických princípov v profesionálom živote znižuje kvalitu života osobného, ale aj profesionálneho.

V tejto eseji najprv uvediem krátky vymyslený príbeh, ktorý poukáže na jednotlivé vyššie uvedené princípy osobnej etiky a neskôr rozoberiem ďalšie otázky súvisiace s problematikou. Mená osôb a organizácií uvedených v tomto príbehu sú vymyslené a ich podobnosť so skutočnosťou je len náhodná.

Príbeh

Peter Lombrandt sa ráno zobudil neskoro, a tak sa musel ponáhľať, aby stihol svoju každodennú rannú kávu. Dnes ho čakal ťažký deň. Jeho firma, ObjectLands Inc. ho vyslala na konferenciu o nových trendoch vo vývoji distribuovaných aplikácií a softvérovom manažmente nazvanú Trends Y2K. Vôbec nechápal, prečo poslali práve jeho a prečo práve v čase, keď sa nasadzoval do praxe projekt (S2000), ktorému už vyše rok a pol šéfoval. Peter sa tam netešil, lebo presne vedel ako to na takých konferenciách chodí – dlhé a úmorné prednášky sa striedajú jedna za druhou a rečníci sú buď pútaví, ale o problematike, o ktorej rozprávajú nemajú ani zdania, alebo sú to ľudia, čo vedia o svojej téme veľa, ale neboli obdarení darom reči. A keby aj prednášky boli akokoľvek pútavé, nemal vôbec chuť počúvať nejakého chytráčika, čo práve vyšiel z MIT, ako mu rozpráva, čo a ako by mal robiť. Tak či tak, čaká ho celodenné „uspávanie hadov“ a najradšej by sa celej konferencii vyhol. Nalial do seba rýchlo horúcu kávu a otrávený zbehol pred svoj dom k autu. Pred dverami svojho nového Buicku si ešte chvíľku v mysli preberal všetky rôzne varianty, ako by sa dalo konferencii vyhnúť. Napadlo mu, že by mohol zavolať Francisovi, že je chorý a požiadať ho, aby na tú konferenciu išiel za neho. Vybral z vrecka telefón, ale skôr, než vytočil Francisovo číslo, celú myšlienku zavrhol a poslal ju na smetisko dejín.

Ešte otrávenejší sadol do auta a vydal sa do centra mesta. Aj cesta autom ho znervózňovala, lebo trasa, ktorou bežne chodieval do práce, bola vždy o takomto čase prejazdná, no E21 do centra mesta bola každé ráno preplnená a na každom kroku sa tvorila zápcha. Peter bol vedúci malého vývojárskeho tímu, ktorý sa venoval tvorbe intranetových riešení na báze Web aplikácií. Tím práve končil projekt informačného systému pre jednu malú lokálnu poisťovňu a hoci projekt nebol veľmi rozsiahly, vliekol sa už príliš dlho. Jeho tím patril vo firme k tým menej podporovaným, lebo oblasť ich práce nepatrila medzi hlavnú špecializáciu firmy. Peter bol postupom projektu poriadne znechutený, lebo vedenie firmy už na neho začínalo čím ďalej, tým viac tlačiť a dávať mu nepríjemné otázky týkajúce sa postupu projektu. Z chmúrnych myšlienok sa vytrhol až na poslednom semafore pred konferenčnou budovou.

V budove vládla typická konferenčná atmosféra. Hromada ľudí sa tlačila pred prezenčným stolíkom a pekná slečna sa spoza neho na všetkých usmievala, rozdávala im účastnícke kartičky a poukazy na obed. Všade rozvoniavala káva a vo fajčiarskej časti vstupnej haly stúpal hustý cigaretový dym. Peter sa rozhliadol, či nezbadá niekoho známeho, ale ani po podrobnom skúmaní nezbadal žiadnu známu tvár. Účastníci sa, ako na každej konferencii, delili na dva tábory. Prvý tábor tvorili vývojári, tí sa dali rozoznať podľa ležérneho oblečenia a veku tak do tridsať. V druhom tábore sa zas nachádzali samí manažéri - oblek, kravata, rolexky na ruke. Peter sčasti patril do oboch táborov a k obom mal rovnako blízko. Bol napoly vývojár, hoci jeho oblečenie ani vek 43 rokov tomu nenapovedali, a napoly manažér. Po prezentácii a dopití druhej dnešnej kávy, vošiel Peter do prednáškovej sály. Sála bola len spolovice plná. Na pódiu bolo rozložených asi 10 počítačov a obrovský projektor. Prvý prednášajúci si ešte narýchlo prezeral svoje poznámky a hneď potom, ako sa zavreli dvere na sále, sa konferencia začala.

Prvá prednáška – Budúcnosť XML, sa venovala XML technológii a jej využitiu pri v intranetových aplikáciách. Prednášateľ bol napodiv pôsobivý i odborne zdatný vo svojej téme. Prezentoval XML ako najnovší trend v spôsobe výmeny údajov a prorocky rozprával o tom, ako onedlho bude XML štandardom, ktorému sa nebude dať vyhnúť. Po úvode do problematiky sa rozprúdila siahodlhá odborná diskusia o XML, do ktorej sa väčšinu zapájali vývojári. Títo ľudia boli plní elánu a nadšenia, ktoré z nich priam sršalo. Peter len ticho pozoroval ich zvedavé otázky a veľmi sa o celú vec nezaujímal. Pripomínali mu seba samého tak pred dvadsiatimi rokmi. Tiež bol plný energie a zaujímal sa o každú novú technológiu, sledoval každý trend a snažil sa byť „in“. V kútiku duše im možno závidel, lebo ich elán a nadšenie mu už dávno chýbali. Neraz zažil projekt, na ktorom sa snažil presadiť novú technológiu, alebo nejaký nový metodologický postup a jeho úsilie skrachovalo. Buď vedenie firmy, v ktorej pracoval, nemalo záujem o použitie nového postupu či trendu, alebo sa neskôr ukázalo, že technológia nie je až taká skvelá, akou sa zdala začiatku a projekt sa nedokončil. Bol v tomto smere skeptikom a nemal záujem učiť sa niečo nové, čo za pár rokov vyjde z módy a zapadne prachom. Zažil už príliš veľa omylov, a tak nechcel zbytočne riskovať.

V hlave sa mu vynorila spomienka na projekt, na ktorom pracoval pred mnohými rokmi ešte ako mladý programátor. Projekt bol určený pre firmu pracujúcu v oblasti likvidácie odpadov – GTC. Peter sa v tom čase zúčastňoval na analýze projektu a štúdii vhodnosti. Projekt zahŕňal riešenie vnútropodnikového informačného systému a po preštudovaní požiadaviek od GTC nasledoval výber technológie a spôsobu implementácie. Hlavný analytik projektu Bart Mahony navrhoval ako jednoznačné riešenie použiť Fox Pro, pretože ObjectLands mala v tom čase s týmto vývojovým prostredím veľké skúsenosti. V tom čase však bolo jasné, že Fox Pro je na ústupe a perspektívosť tohto riešenia je nízka. Peter sa snažil poukázať na nezmyselnosť tohoto výberu, lebo bolo viac ako pravdepodobné, že po roku vývoja produktu (to bola odhadovaná doba trvania projektu) dostane zákazník namiesto kvalitného riešenia zastaralú relikviu. Jeho pripomienky však neboli brané do úvahy, nakoľko Mahony patril medzi významné osobnosti firmy a tiež znášanie kritiky nepatrilo medzi jeho silné stránky. S Mahonym mal Peter už v minulosti niekoľko konfliktov a dalo by sa povedať, že sa nemali radi. Bolo mu navyše vedením naznačené, že podobné návrhy a pripomienky by mal v budúcnosti lepšie zvážiť skôr, než ich prednesie.

Pre neho to bol úder pod pás, ktorý nečakal. Mal chuť celú vec vykričať na verejnosti, či povedať o tom ľuďom z GTC, ale nenašiel potrebnú odvahu. Rezignoval. Od toho momentu, ktorému predchádzalo ešte niekoľko podobných príhod, sa prestal zaujímať o softvér, na ktorom pracoval. Jeho dovtedajší prístup k práci, jeho nasadenie, záujem o všetko nové a perspektívne sa vytratili. Prácu už bral viac ako zamestnanie, a nie ako hobby. Nevidel vo svojom snažení zmysel, lebo hoci sa snažil produkovať kvalitný kód, dokumentáciu a navyše aj pomáhať pri tom iným, jeho okolie ho vnímalo ako naivku a v tomto smere nedostával nijakú pozitívnu spätnú väzbu. Rozčuľoval sa nad postupom niektorých firemných projektov. Nechápal laxný prístup niektorých svojich kolegov pri vývoji softvéru. Vo firme neexistovalo žiadne testovacie pracovisko a preto sa mnohokrát softvér testoval až v mieste nasadenia. Projektové stretnutia, na ktorých by sa diskutovalo o postupe projektu, či dokonca by sa prezeral a posudzoval hotový kód, boli pre Petra len nesplneným snom. Kvalita produkovaného softvéru bola skutočne nízka, kvalitná analýza a špecifikácia boli len abstraktnými pojmami. Niekoľkokrát sa pokúsil z firmy odísť, lebo pracovná atmosféra a praktiky vo firme sa priečili jeho presvedčeniu, no odísť nebolo také ľahké. Postupne sa dostal v hierarchii podniku do vyšších sfér a tak sa zmieril s tým, že v ObjectLands zostane. Zmieril sa s praktikami firmy a stratil záujem o čokoľvek nové, čo by so softvérom súviselo.

Zo spomienok ho vytrhol záverečný potlesk pre prvého prednášajúceho. Okolosediaci vyzerali vyčerpaní a nesústredení a to ich ešte čakala jedna prednáška, skôr než sa pôjde na obed. Druhá prednáška mala názov „Ako vylepšiť proces vývoja softvéru.“ a prednášateľom bol profesor Wilrowski z miestnej univerzity. Profesora osobne poznal a aj téma sa mu pozdávala, tak sa rozhodol, že si ho vypočuje. Úvod prednášky bol venovaný výpočtu rôznych chýb a nedostatkov, ktoré súvisia s vývojom softvérového produktu. Boli spomenuté problémy ako nedostatočná

komunikácia v projektových tímoch, slabá dokumentácia, netransparentnosť postupu projektu a iné. Petra téma ozaj zaujala a prednášku si vypočul až do konca. Po prednáške sa všetci odobrali na obed a tam mal čas pouvažovať o tom, čo počul.

Keď sa zamyslel nad celým obsahom prednášky, mal dojem, že to v jeho firme reálne nemôže fungovať. Veď len taká dokumentácia. V ObjectLands sa na projekte dokumentovali len tie najnevyhnutnejšie veci. Používateľská príručka a nejaká tá technická dokumentácia síce vždy vznikli, no nikto tomu neprikladal veľký význam. Nikdy na to nebol čas. Akokoľvek dobre sa snažili odhadnúť dobu trvania projektu, takmer vždy sa projekt nedokončil na čas. Záver projektov bol vždy v znamení 12 až 14 hodinových pracovných maratónov a boja s časom. Manažmnet ObjectLands mal len jediný záujem: dokončiť každý projekt za čo najkratší čas a pri najmenších nákladoch. Nad svojou konkurenciou firma víťazila vždy cenovou ponukou a rýchlosťami dodávok, hoci kvalita výsledných produktov bola viac než chabá. Komentovanie kódu vonkoncom nespĺňalo ISO normu (25% komentárov v kóde) a ošetrovanie a testovanie produktov boli tiež len okrajové aktivity. Firma dostáva denne niekoľko desiatok sťažností a dokumentov o objavených chybách, ale to nikoho neprekvapuje. Namiesto toho, aby chyby vo výrobkoch boli odstránené už v prvopočiatku, ráta sa hneď s tým, že prax na chyby poukáže sama.

Z prednášky mu v hlave najviac utkvela jedna pasáž: „Vaša organizácia sa musí zmeniť na učiacu sa organizáciu. Neustále učenie sa a vyučovanie nových trendov a technológií, musí byť zakomponované priamo do pracovného procesu. Vaši zamestnanci musia dostať v tomto smere patričnú voľnosť, aby mohli prichádzať sami s novými myšlienkami a nápadmi. Len tak bude pracovný proces úspešný.“ Peter si len povzdychol, lebo vo svojej doterajšej praxi sa s ničím podobným nestretol. Radoví zamestnanci ObjectLands nikdy nemali dostatočnú voľnosť. Ich práca bola buď priamočiaro nalinkovaná, alebo častejšie sa stávalo, že sa nik nestaral o to, ako sa pracuje. Chýbala motivácia, a tak nik nepotreboval vedieť niečo nové.

Obed sa už dávno skončil, no Peter zostal zamyslene sedieť v reštaurácii. Po chvíli sa zdvihol a pobral sa späť do prednáškovej sály. V tom zazvonil jeho mobilný telefón. Bol to Petrov priamy nadriadený, John Malcom. „Ahoj Peter, máme problém. V S2000 sú zas tie problémy so synchronizáciou. Volali mi z Prvej Národnej, že sa im to celé zrútilo a firma preto stojí. Príď okamžite do práce, lebo tí tvoji mamľasi [tím] určite nemajú ani zdania ako sa tá chyba vyskytla a už vôbec nie, ako ju odstrániť. Sadni okamžite do auta, lebo z toho bude vážny problém!“. Peter ubezpečil Johna, že okamžite príde, a z konferencie odišiel. V duchu zanadával. Veď Johnovi niekoľkokrát opakoval, že S2000 nie je poriadne otestovaný, a že predpokladá problémy. John ho vtedy ubezpečil, že sa nič hrozného nemôže stať a že projekt sa musí odovzdať, lebo silnejú tlaky a všetci majú enormný záujem na odovzdaní S2000. A teraz bude na vine on. Ešte raz v duchu zanadával. Cestou v aute vytočil číslo svojej ženy Simon a oznámil jej, že dnes asi tak skoro z práce nepríde, lebo má nejakú súrnu robotu. Neuniklo mu, že Simon si pri telefonáte niekoľkokrát vzdychla. Spomenul si, ako jej minulý týždeň sľúbil, že pôjdu do

divadla. Vedel, že z toho tento a určite ani ďalší týždeň nič nebude. Uvedomil si, že chronicky nestíha. Ani si poriadne nevedel spomenúť, kedy mal naposledy príjemný víkend len s ňou. Ešte aj z dovolenky pred pol rokom ho odvolali v polovici, lebo niečo nechodilo. Znova, ako viackrát pred tým, ho premohol pocit, že by radšej robil niečo úplne iné. Zatúžil po niečom pokojnejšom, menej stresujúcom ako napríklad ..... hmmmm záhradkár?

O príbehu

Príbehom o Petrovi Lombrandtovi som chcel poukázať na jednotlivé body etického kódexu, ktoré sa týkajú vzťahu softvérového inžiniera k sebe a ich časté porušovanie v praxi. V príbehu sa vyskytlo niekoľko momentov a faktov, ktoré sú v rozpore s ôsmym bodom ACM a IEEE etického kódexu. Medzi hlavné momenty patria tieto:

Petrova strata záujmu

Príčin, pre ktoré Peter stratil záujem rozširovať svoje vedomosti v oblasti svojej profesie (svojím správaním porušoval body 8.01 až 8.05) mohlo byť niekoľko, no asi najvýznamnejšou príčinou je zlá atmosféra v ObjectLands a ich brzdenie rozvoja svojich zamestnancov. Vedenie ObjectLands, hoci možno nevedome, rozhodne znemožňovalo svojim zamestnancom dodržiavanie etického kódexu. V príbehu vyslovil profesor Wilrowski dva závažné postrehy, ktoré sú v tejto súvislosti podstatné [2]:

- Organizácia musí podporovať svojich zamestnancov v neustálom napredovaní a ich osobnom vzdelávaní.
- Vzdelávanie musí byť prirodzene zakomponované do pracovného procesu.

Je prirodzené, že jedinec sa začne správať neeticky, ak sa celé jeho okolie správa rovnako. Riešením Petrovho problému by bol odchod z ObjectLands do inej organizácie, čo by vzhľadom na jeho osobu malo etický aj profesionálny prínos. Peter si uvedomoval neprofesionálnosť svojho správania (v súlade s bodom 8.09) i správania sa firmy, no osobné a iné dôvody ho viedli k tomu, že ObjectLands neopustí.

V profesii softvérového inžiniera nie sú doteraz zavedené jasné explicitné pravidlá a požiadavky na vzdelávanie sa jednotlivca. Ak má byť cieľom kvalitný, bezpečný a spoľahlivý produkt, musí sa aj jednotlivec neustále zdokonaľovať, tak ako sa mení jeho okolie. Pekným porovnaním v tomto smere je lekárska profesia. Napríklad chirurg je vo svojej profesii neustále kontrolovaný svojím okolím a jeho úspech úzko súvisí s rozširovaním jeho znalostí. Ak chce byť úspešný, musí prejsť

sériou atestácií a náročných zákrokov. Lekárska profesia núti jednotlivca, aby sa neustále vzdelával a rozširoval si obzor a vedomosti, a pritom mu vytvára k tomu náležité možnosti. Preto je dôležité, aby boli softvéroví inžinieri oboznámení so znením tohto kódexu a usilovali sa o jeho dodržiavanie.

Analytik Mahony

Analytik projektu GTC tiež niekoľkokrát porušil etický kódex. Jeho jednoznačné odsúdenie Petrovho návrhu na zmenu postupu v projekte je v rozpore s bodom 8.07 (*Nezaobchádzať s nikým nespravodlivo z dôvodu svojich predsudkov voči danej osobe*). Podľa môjho názoru je bohužiaľ podobné správanie v praxi dosť bežné. S predsudkami na pracovisku sa môžeme stretnúť v nejednej firme. Či už sa jedná o predsudky rasové, profesionálne, osobné alebo akékoľvek iné, správny profesionál by sa mal vedieť nad ne povzniesť a vyhnúť sa im. Ani Mahonyho postup v súvislosti s projektom GTC nebol celkom v súlade s princípmi etiky. Mahony sa neeticky správal vo vzťahu ku svojmu zákazníkovi, lebo vedome chcel použiť pri riešení projektu zastarávajúcu technológiu.

Nadriadený John Malcom

Malcom prinútil Petra, aby odovzdal projekt S2000 aj napriek tomu, že bol upozornený na jeho nedostatky. Dostal sa tak do rozporu s bodom 8.08 etického kódexu - *Nenavádzať nikoho na porušovanie tohto kódexu*. Môžeme povedať, že i firma ObjectLands ako celok tento bod kódexu porušovala. Podľa môjho názoru neexistuje jednoznačný návod, ako sa zachovať v prípade, že sa zamestnanec stretne s podobnou formou navádzania na porušenie etiky. Pragmatickým riešením takejto situácie by bolo odmietnuť vykonať požadované činnosti, čo by však veľmi pravdepodobne prinieslo spätnú negatívnu odozvu od žiadateľa. Jednoznačné riešenie neexistuje a závisí značne od konkrétneho prípadu. Nemožno čakať od zamestnanca, ktorému hrozí prepustenie z práce ak nevykoná neetickú činnosť, že sa tejto práce hneď automaticky vzdá len preto, lebo je to etické. Tento problém je potrebné vnímať v širších súvislostiach (pozri prípad Cindy Yardley [3]). V takomto prípade je ale podstatné aspoň vnímať porušovanie etického princípu a uvedomiť si jeho nezlučiteľnosť s profesionalitou (bod 8.09). Ďalej Malcom porušil aj princíp 8.07 tým, že sa nevyhol predsudkom vo vzťahu k Petrovmu tímu.

Manželka Simon

Peter mal pocit, že popri svojej práci Simon zanedbáva. Bolo to v dôsledku nadmerného pracovného zaťaženia a neustálych stresov. Organizácie ako ObjectLands často preťažujú svojich zamestnancov a nútia ich k maximálnym

výkonom. Tento prístup má však minimálne dva negatívne dôsledky. Preťažovaním zamestnanca výrazne klesá kvalita jeho života a on preto často stráca záujem o profesionalitu, podáva menej kvalitný výkon. V tomto prípade bolo neetickým správanie sa zamestnávateľa (ObjectLands) voči svojmu zamestnancovi (Peter).

Záver

Na záver je potrebné dodať, že chápanie etiky softvérového inžiniera je u nás i vo svete v plienkach. Je preto potrebné vyvinúť mnoho úsilia, aby sa dodržiavanie etických princípov stalo bežným javom v softvérovej profesii tak, ako je to bežné v profesiách iných. Je potrebné, aby hlavne komerčné organizácie vytvárali pre svojich zamestnancov také prostredie, aby ich hlavným pracovným cieľom bola čo najvyššia profesionalita. Je tiež nutné znížiť zaťaženie a stres jednotlivca v pracovnom procese, lebo sa môže stať, že tu bude zrazu príliš veľa záhradkárov.

Použitá literatúra

[1] D. Gotterbarn, K. Miller, S. Rogerson: Computer Society and ACM Approve Software Engineering Code of Ethics. IEEE Computer, Vol. 32, No. 10, October 1999, Volume 32, str. 84-88.

[2] Noel M. Tichy, Eli Cohen: The techaing organisation. IEEE engeneering management review, Winter 1998, str. 79.

[3] Richard G. Epstein: The case of killer robot, Article-8: Silicon Techtronics employee admits faking software tests. West Chester University of PA, CWRU 1998.

Uf, pomyslel si, tak toto bolo trefné. Veď aj on sám je príkladom toho, že osobný rozvoj a nutnosť rozširovania vedomostí sa stali bežnou praxou. Počas jeho práce programátora bolo úplne bežné, že vedomosti a zručnosti programátorov sa v pravidelných intervaloch overovali. On sám bol „odložený“ do tohto inštitútu preto, lebo nesplnil požadované kritériá. Spomínal si, ako ich vo firme preskúšavali náročnými testami a nútili ich tak udržiavať sa na špici v technologickom rozvoji. Tie testy pripomínali skúšobné previerky pilotov a kozmonautov. Nebolo to vôbec príjemné, keď si človek uvedomil, že môže ľahko prísť o prácu a navyše elektródy, pomocou ktorých ho pripájali do systému ho vždy neuveriteľne štípali. Na druhej strane sa však výrazne zlepšil prístup programátorov ku svojej profesii a mnohí „takzvaní programátori“ museli profesiu opustiť. Dokonca počul, že dnes je už bežnou praxou, že programátora nad 35 rokov automaticky preraďujú na menej náročné a kreatívne úlohy, lebo bol dokázaný významný pokles rozvoja softvérových zručností po tomto veku. Nostalgicky si zaspomínal na svoje staré programátorské časy, ale vedel, že by sa už nechcel k tomu vrátiť. Asi by už nestíhal to tempo, ktoré

musí byť niekoľkonásobné oproti tomu v jeho programátorskej ére. Zavrel poslednú esej a rozhodol sa, že ešte napíše šéfovi správu o prečítaných materiáloch. Loyd si na to hrozne potrpel.

Záver

Wesly Woodword mal po prečítaní esejí možnosť porovnať posun a rozvoj informačných technológií a softvéru vôbec za 50 rokov a s tým spojenú zmenu v etických problémoch tejto profesie. Možno povedať, že takmer s určitosťou sa v budúcnosti udejú zmeny, ktoré budú znamenať nové chápanie etických pravidiel a v niektorých prípadoch dokonca spôsobia, že niektoré body etického kódexu už nebudú opodstatnené. Určité je však aj to, že niektoré princípy ostanú naďalej v platnosti a ich dodržiavanie bude naďalej spojené len s etikou a profesionalitou softvérového inžiniera. Týmito princípmi sú:

1. *Vzťah a zodpovednosť softvérového inžiniera ku spoločnosti*
2. *Vzťah softvérového inžiniera ku svojej profesii*
3. *Vzťah ku sebe samému*

Uff. Po dlhom dni Wesly cítil, že jeho mozgový okysličovač sa začína prehrievať. Narýchlo vydal príkaz na prenos správy k Loydovi a odpojil svoj nervový systém zo siete. Konečne prišiel ten krásny pocit uvoľnenia, keď nemusí kontrolovať tok svojich myšlienok z obavy odpočúvania. Zo stola si zobral pokrčené vydanie svojich obľúbených novín, narýchlo ich vopchal pod pazuchu a nechal sa iónovým detonátorom preniesť domov...

Softvérové procesy

Znepríjemňujú nám život, a predsa sa bez nich nezaobídeme

Tím SOWA

Pred čítaním odborných článkov

Po ľahkom, priam populárno-náučnom, štýle esejí na etický kódex prejdime k niečomu menej stráviteľnému. Áno, sú to softvérové procesy. Nočná mora, ktorej sa pokúšame my, (budúci) softvéroví inžinieri zbaviť a predsa ich používame takmer na každom kroku svojej práce. Nasledujúca časť publikácie sa Vám pokúsi vniesť troška svetla aj do tejto problematiky.

Prvý článok je stručným pohľadom na to, čo softvérové procesy vlastne sú a podrobne opisuje najznámejšie procesné modely vývoja softvéru. Ďalšie články sú venované novým trendom v oblasti procesov vývoja. Ako prvým je hypergrafový model, ktorý je formálnym zovšeobecnením známeho evolučného modelu. Pri súčasnom rozmachu Internetu a s ním spojenej prezentácie informácií prostredníctvom webu, sme nemohli obísť ani tematiku softvérových procesov pri tvorbe web aplikácií, ktorú ešte aj dnes samotný tvorcovia aplikácií veľmi podceňujú. Väčšinou každého zaujíma len vlastné ihrisko a preto aj vývojári sa najviac zaujímajú práve o to svoje - implementáciu. Extrémne programovanie je model, ktorý napomáha vytvárať flexibilné programy, reagujúce na časté zmeny v požiadavkách v čo najkratšom čase a za vynaloženia najmenších prostriedkov. No a nakoniec tu máme Unified proces, ktorý nám poskytuje iteratívny pohľad na vývoj softvéru a návod, ako efektívne používať rozšírený jazyk pre modelovanie vývoja softvérových systémov (UML).

Takže smelo obráťte na ďalší list a príjemné čítanie.

Úvod do softvérových procesov.

Peter Liczki, Michal Kucbel

Tvorba softvéru v softvérových spoločnostiach so sebou prináša veľké množstvo súčasne prebiehajúcich procesov. Viaceré z nich sa však netýkajú priamo softvérového inžinierstva. Ako príklady môžeme uviesť ekonomické procesy, sociálne procesy alebo procesy školení.

Procesy zaoberajúce sa technickými záležitosťami a manažmentom vývoja softvéru sa nazývajú *softvérovými procesmi.*

Softvérový proces môžeme charakterizovať nasledovne [3]:

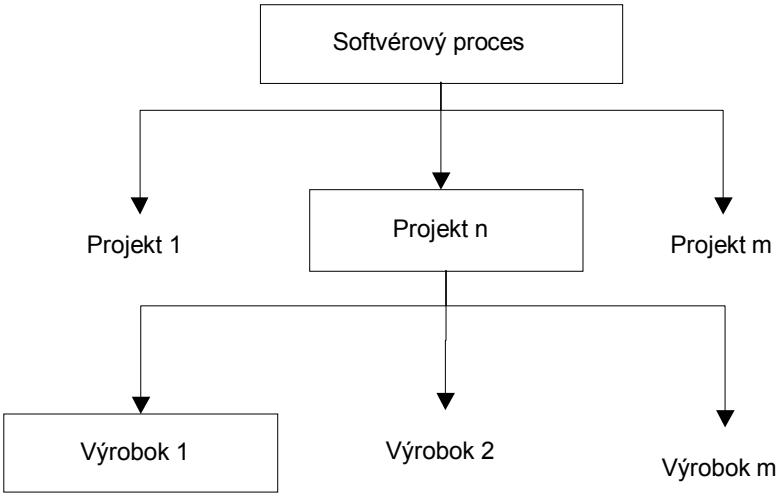
- Predpisuje všetky hlavné aktivity, ktoré treba vykonať.
- V procese sa ráta so zdrojmi ako s množinou obmedzení na činnosti (istý rozvrh).
- Proces môže pozostávať z viacerých podprocesov, ktoré sú istým spôsobom prepojené. Proces môže byť zadefinovaný ako hierarchická štruktúra podprocesov a každý podproces môže mať vlastný procesný model
- Každá činnosť procesu má zadefinované začiatočné a koncové kritérium. Presne sa dá potom určiť kedy daná činnosť začína a končí.
- Činnosti sú zoradené do sekvencie, takže je jednoduché určiť kedy sa vykonáva jedna činnosť vzhľadom k druhej.
- Každý proces sleduje istú množinu princípov, ktoré vysvetľujú ciele každej aktivity.
- Obmedzenia alebo riadenie môžu vplývať na činnosť, zdroj alebo produkt. Napríklad rozpočet a rozvrh môžu skrátiť čas trvania istej činnosti alebo pomocné nástroje môžu ovplyvniť spôsob akým sa daný zdroj bude využívať.

Softvérový proces spolu so softvérovým projektom a softvérovým výrobkom sú pojmy, ktoré sa veľmi často opakujú v softvérovom inžinierstve. V ďalšom si zadefinujeme vzťah medzi nimi. Vychádzame pri tom z [1] a [2]

Softvérový proces, ako to bolo uvedené vyššie, určuje postup vývoja softvéru a manažment vývoja softvéru. Softvérový projekt je naproti tomu vývojový projekt, v ktorom sa softvérový proces využíva. Softvérové

produkty sú výstupom zo softvérového projektu. Každý softvérový projekt začína požiadavkami na výsledný softvér a (vo väčšine prípadov) končí softvérovým produktom, ktorý uspokojuje tieto požiadavky. Softvérový proces určuje teda iba abstraktnú množinu činností, ktoré sa majú vykonať pri vývoji výrobku.

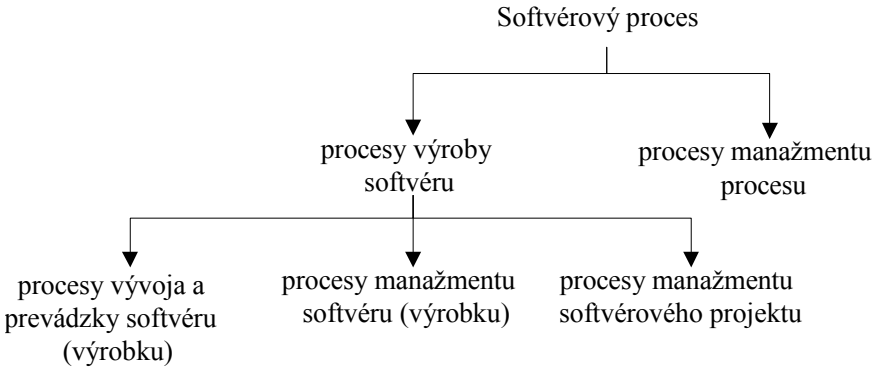
V podstate si môžeme softvérový proces predstaviť ako istý abstraktný typ a každý projekt, ktorý využíva tento proces za inštanciu daného typu. Z toho vyplýva, že môže existovať viacero projektov, ktoré využívajú ten istý proces, čo zachytáva aj obr. 1.



Obr. 1. Procesy, projekty a produkty [1] a [2]

Proces teda treba zakaždým prispôsobiť podmienkam toho ktorého projektu (presnejšie popísať postup). Tento priestor vypĺňajú plány projektov.

Každý z už spomínaných pojmov proces, projekt a produkt si vyžaduje rozdielne (vlastné) procesy. Na obr. 2 je uvedené rozdelenie softvérových procesov práve z tohto pohľadu. Procesy vývoja a prevádzky softvéru a procesy manažmentu softvéru sa zaoberajú produktom, procesy manažmentu softvérového projektu sa venujú projektu a procesy manažmentu procesu sa zaoberajú samotným procesom.



Obr. 2. Klasifikácia softvérových procesov [1] a [2]

- **Proces vývoja a prevádzky softvéru**

Hlavným procesom zaoberajúcim sa produktom je *Proces vývoja a prevádzky softvéru*, ktorý sa zaoberá produkciou požadovaného produktu (programu) a ostatných produktov (napríklad používateľskej príručky alebo špecifikácie požiadaviek). Pokrýva činnosti priamo spojené s vývojom softvéru – špecifikácia požiadaviek, architektonický návrh, podrobný návrh, implementácia, integrácia a testovanie, prevádzka a údržba. Hlavným cieľom tohoto procesu je vyvinúť produkt, ktorý uspokojí zákazníka.

- **Procesy manažmentu projektu**

Neoddeliteľnou súčasťou vývoja softvéru je aj vhodné riadenie. Na to aby sa dodržala požadovaná akosť vytváraného softvéru (stanovená cena, stanovený čas na vývoj a splnenie požiadaviek), je nevyhnutné, aby sa potrebné zdroje vhodne vyhradili pre každú činnosť a sledoval postup jednotlivých činností a v prípade potreby zabezpečili korektúry. Všetky tieto aktivity spadajú medzi *procesy manažmentu projektu.*

- **Procesy manažmentu softvéru**

V priebehu vývoja softvéru sa návrh, zdrojové kódy a aj špecifikácia často menia. Navyše sa tieto zmeny môžu vyskytnúť v hociktorom čase životného cyklu softvéru. Práve *Procesy manažmentu softvéru* sa zaoberajú systematickým riadením zmien, ktoré sa vyskytnú počas vývoja.

- **Procesy manažmentu procesu**

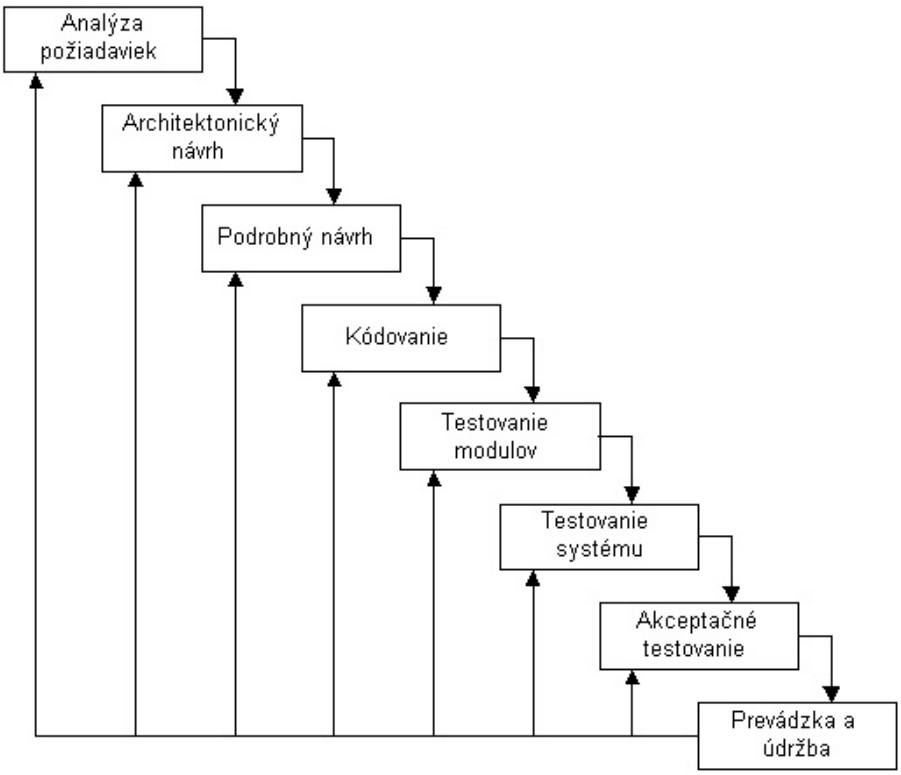
Softvérový proces si nepredstavujme ako niečo statické, nemenné. Vynakladá sa veľké úsilie na to, aby sa vhodnou zmenou existujúcich procesov, docielil proces, pomocou ktorého by sa mohol vyrábať softvér vyššej akosti. Softvérové procesy sa neustále vylepšujú a tieto zmeny riadia *procesy manažmentu procesu.*

Procesy sa teda neustále vyvíjajú a z toho dôvodu je potrebné aby, sa získané skúsenosti uchovávali a prípadne predávali iným. Na opis procesov existuje viacero spôsobov: textovo, obrázkom alebo ich kombináciou. Takto opísané procesy predstavujú modely, ktoré zachytávajú najdôležitejšie funkcie daného procesu.

Opisu niektorých najznámejších procesných modelov vývoja softvéru sú venované aj nasledujúce časti. Začneme opisom tzv. klasických procesných modelov ako vodopádový model, evolučný model, formálna transformácia a špirálový model. Za tým nasledujú príspevky, v ktorých sa zameriame na novšie procesné modely.

Vodopádový model

Vodopádový model je jedným z najdlhšie používaných modelov v procese vývoja softvérových systémov a bol odvodený z iných inžinierskych procesov. Poskytuje možnosť čo možno najviac sprehľadniť proces vývoja, pretože pri tomto prístupe k vývoju sotftvéru sa striktne oddeľujú jednotlivé časti vývoja do samostatných fáz. Prechod do novej fázy je možný len po úplnom dokončení prác týkajúcich sa súčasnej fázy vývoja. V opačnom prípade by mohla vzniknúť potreba dodatočne sa vrátiť do predchádzajúcej fázy a takýto postup by nebol pri použití tohoto modelu správny.



Obr. 3. Schéma vodopádového modelu [1][3][4]

Základné fázy vývoja sú (pozri obrázok 3):

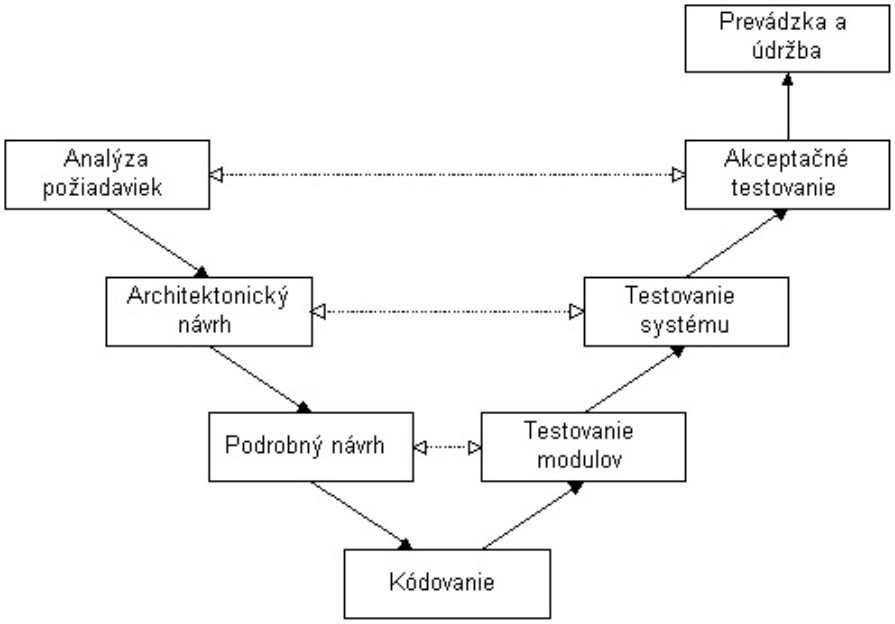
1. *Analýza požiadaviek:* v spolupráci so zákazníkom sa určia ciele projektu, požadované funkcie vytváraného systému a prípadné ohraničenia. Získané údaje sa zdokumentujú spôsobom zrozumiteľným aj pre zákazníka aj pre tvorcov systému.
2. *Architektonický a podrobný návrh:* systémový návrh delí požiadavky pre softvérové alebo hardvérové systémy a vytvára celkovú architektúru systému. Softvérový návrh obsahuje návrh funkcií softvérového systému.
3. *Implementácia("kódovanie") a testovanie modulov:* v tejto fáze sa softvérový návrh transformuje do programov alebo programových modulov. Testovanie modulov znamená overenie, či implementácia každého modulu zodpovedá špecifikácii.
4. *Integrácia a testovanie systému:* jednotlivé programy a moduly sa v tejto fáze integrujú a následne sa testujú už ako celý systém. Po uistení sa, že systém spĺňa špecifikované požiadavky môže nasledovať distribúcia alebo odovzdanie systému zákazníkom.

5. *Prevádzka a údržba:* táto fáza predstavuje samotné používanie systému v praxi. Údržba v sebe zahŕňa odstraňovanie chýb, ktoré neboli objavené pri tvorbe systému, vylepšovanie modulov systému a pridávanie nových funkcií do systému.

V praxi je však dosť ťažko jednotlivé fázy presne takto oddeliť z dôvodu ich úzkej zviazanosti. Každá fáza závisí od informácií, ktoré sú výstupom z predchádzajúcej fázy. Rovnako aj chyby, ktoré sa urobili v niektorej fáze, sa väčšinou odhalia až v ďalších fázach. Posledná fáza môže sama obsahovať niektoré alebo aj všetky predošlé fázy vývoja, ktoré sa môžu v cykloch opakovať tak ako sa opakujú požiadavky na rozšírenie systému prípadne na odstraňovanie chýb.

Najväčšou výhodou tohoto modelu je dobrá viditeľnosť postupu prác na projekte čo je zabezpečené presným definovaním hraníc medzi jednotlivými fázami vývoja ako aj definovaním výstupov jednotlivých fáz. Dobrá viditeľnosť uľahčuje riadenie takéhoto projektu, ale aj spoluprácu so zákazníkom. Naopak nevýhodou modelu je jeho veľká miera abstrakcie, ktorá neumožňuje efektívne ho použiť pre väčšie projekty. Keďže bol tento model prebratý z iných inžinierskych disciplín, predstavuje skôr fabrický prístup k vývoju softvéru. Nezohľadňuje skutočnosť, že softvér rieši väčšinou úplne nové problémy, a preto jeho tvorba prebieha vo viacerých iteračných cykloch.

Úpravou tohoto modelu vznikol tzv. **V-model** (pozri obrázok 4), ktorý navyše znázorňuje ako súvisia testovanie, analýza a návrh. Model má tvar písmena V, kde na ľavej strane je analýza a návrh a na pravej strane je testovanie. Tvorba samotného programu je presne v strede. Ľavú stranu diagramu môžeme nazvať aj vetvou vývoja systému a pravú stranu vetvou overovania a používania systému. Prepojenie medzi vetvami (na obrázku znázornené čiarkovane) znamená, že ak sa vo vetve overovania objavia chyby, potom sa musí projekt vrátiť do vetvy vývoja. Presnejšie projekt sa vráti do tej konkrétnej fázy vývoja, ktorá je prepojená s fázou overovania, v ktorej bola chyba zistená.



Obr. 4. Schéma V-modelu [2]

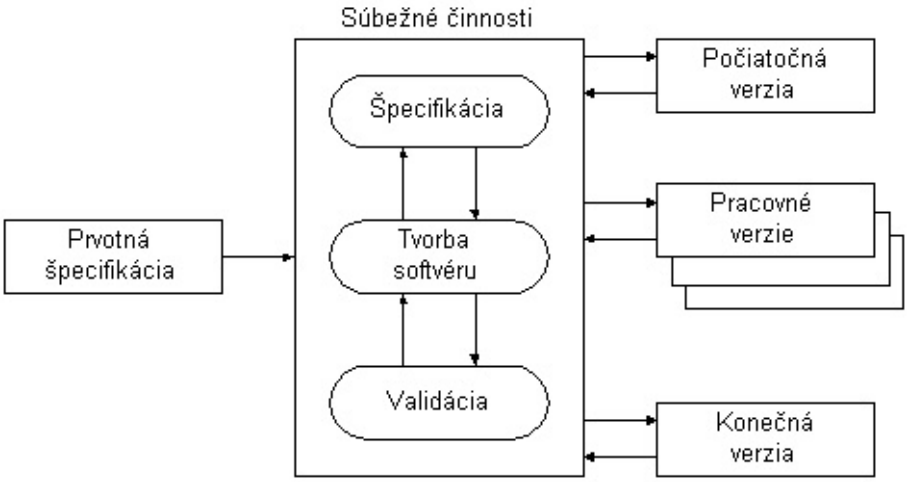
Chyby zistené pri testovaní modulov znamenajú aj potrebu overenia podrobného návrhu. Podobne je testovanie celého systému zviazané s návrhom architektúry systému. Akceptačné testovanie, ktoré by mal vykonávať zákazník, vyhodnocuje splnenie zadaných požiadaviek. Tieto vzťahy sú na obrázku znázornené čiarkovane. Plnými čiarami je znázornený postup v procese vývoja projektu za predpokladu, že sa nevyskytnú žiadne chyby. Pre tento prípad je V-model zhodný s vodopádovým modelom.

Vodopádový model vo svojej základnej podobe tvorí základ mnohých zložitejších modelov vývoja. Samotný model je má viacero nedostatkov, pre ktoré nie je možné ho použiť pri komplexnejších projektoch. Jednu z najväčších nevýhod vodopádového modelu, neuvažovanie cyklov v procese vývoja, sa V-model snaží odstrániť. V-model presne definuje, do ktorej etapy vývoja sa treba vrátiť, ak zlyhá overovanie.

Evolučný model

Použitie tohoto modelu pri vývoji vytvára presne opačnú situáciu ako pri vývoji podľa vodopádového modelu. Tento prístup odstraňuje presné hranice medzi špecifikáciou, tvorbou softvéru a validáciou. Pred vytvorením konečného produktu sa

vytvára čo možno najviac prototypov, ktoré sa konzultujú so zákazníkmi. Prvotný prototyp sa vytvára podľa abstraktnej špecifikácie. Špecifikácia sa pomocou prototypov a vstupov od zákazníka neustále spresňuje až kým sa neuspokoja požiadavky zákazníka.



Obr. 5. Schéma evolučného modelu [4]

Podľa spôsobu, akým sa postupuje pri spresňovaní špecifikácie môžeme deliť tento spôsob vývoja na dva typy:

1. *Prieskumné prototypovanie:* prototypovanie sa začína od častí systému, ktoré sú dobre pochopené. Ďalšie časti systému sa do prototypov zahŕňajú až keď sa vykonzultujú so zákazníkom. Tento typ evolučného modelu je vhodný, keď si za cieľ projektu kladieme získanie presných používateľských požiadaviek, na čo využívame vytvárané prototypy, ale súčasne sa požaduje vytvorenie funkčného konečného produktu pre zákazníka. Pri takomto postupe spresňovania zákazníckych požiadaviek sa predpokladá, že pri prototypovaní dobre ujasnených požiadaviek sa môžu vysvetliť aj menej presné požiadavky.
2. *Vytváranie prototypov na zahodenie:* pri tomto spôsobe prototypovania sa zameriava na hlavne na problematické časti špecifikácie, ktoré potrebujú spresnenie. Prototypovaním sa spresnenie špecifikácie urýchľuje. Tento postup je preto vhodný na vylepšenie špecifikácie požiadaviek na vytváraný systém a vyjasnenie požiadaviek.

Výsledkom procesu vývoja podľa evolučného modelu môže byť buď posledná verzia, alebo upravená špecifikácia, ktorá sa následne použije ako základ vo vývoji

daného produktu, ale už podľa iného modelu, ktorý zaručí vyššiu spoľahlivosť, udržiavateľnosť a robustnosť produktu.

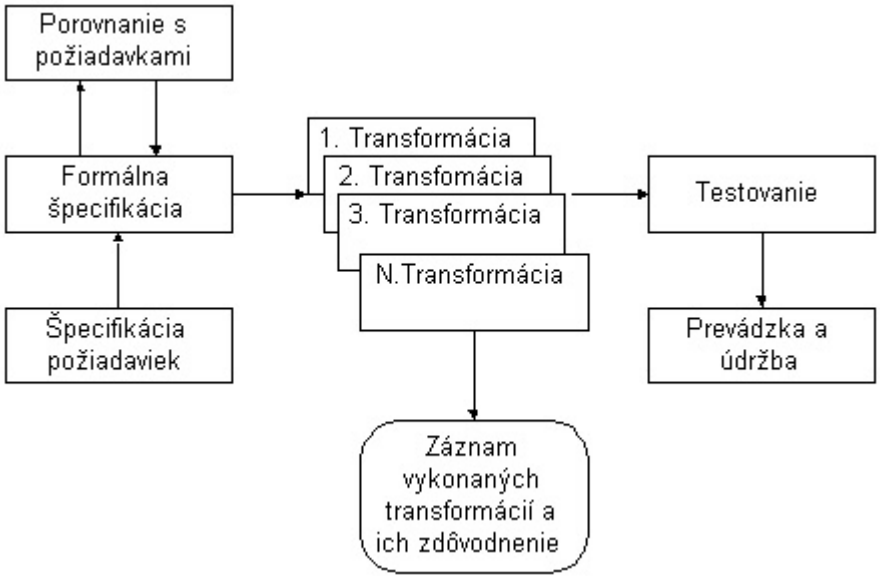
Výhodou pri tomto postupe vývoja je možnosť prispôsobovať sa zmenám požiadaviek, preto sa využíva hlavne v prípadoch, kedy nie je možné na začiatku vývoja vytvoriť dostatočne presnú špecifikáciu, alebo ak vieme, že sa môžu požiadavky počas vývoja výrazne meniť. Efektívne možno tento model použiť pri tvorbe systémov, kde samotný vývoj sa musí rýchlo prispôsobovať aktuálnym požiadavkám. Tento model v sebe zahŕňa aj tri veľké riziká:

1. Zlá viditeľnosti vývoja samotného procesu. Zlá viditeľnosť je spôsobená vykonávaním procesu ako cyklu špecifikácie, tvorby systému a validácie, pričom počet iterácií potrebných na dokončenie systému nie je nikdy vopred známy.
2. Riziko zle štruktúrovaného výsledného produktu. Toto riziko vyplýva z toho, že postup prác na systéme určuje vlastne zákazník, a preto môže byť výsledná štruktúra systému zlá. Zlá architektúra systému môže nakoniec vývoj sťažiť a predražiť.
3. Potreba špeciálnych znalostí. Efektívne vytváranie nových prototypov si vyžaduje od personálu mať určité špeciálne znalosti alebo zručnosti. Z dôvodu týchto problémov sa tento model nepoužíva pri tvorbe veľkých systémov.

Evolučný model možno efektívne uplatniť len pri vytváraní relatívne malých systémov, systémov s obmedzenou životnosťou alebo častí väčších systémov bez presnej špecifikácie. V prípade malých systémov môže byť požiadavka na zmenu systému riešená aj vytvorením celého systému odznova. Vývoj častí väčších systémov podľa tohoto modelu sa týka hlavne používateľských rozhraní alebo modulov umelej inteligencie.

Formálna transformácia

Tento prístup sa snaží eliminovať riziko možných chýb. Celý projekt je založený na formálnej matematickej špecifikácii vytváraného systému, ktorej správnosť je overená. Táto špecifikácia sa následne s pomocou rôznych matematických metód použije pri transformácii týchto požiadaviek do programu. Používané matematické metódy pritom zachovávajú správnosť zaručovanú špecifikáciou, a preto produkt vždy zodpovedá špecifikácii.



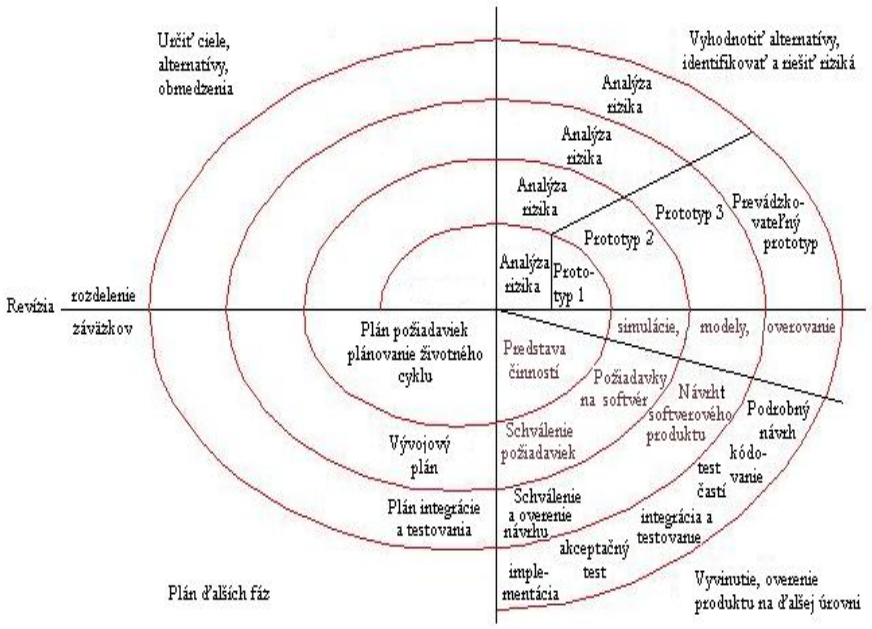
Obr. 6. Schéma modelu formálnej transformácie [3]

Základné kroky pri transformácii v sebe zahŕňajú úpravu dátových reprezentácií, výber algoritmov, optimalizáciu a kompilovanie. Na rozdiel od väčšiny modelov, formálna transformácia presne určuje ako postupovať pri vytváraní produktu z danej špecifikácie. Slabým miestom je ale práve potreba presne vytvorenej formálnej špecifikácie, bez ktorej sa nemôže formálna transformácia uskutočniť.

Veľkou výhodou tohoto modelu je možnosť automatizácie celého softvérového produktu. Najväčšia časť úsilia pri takomto postupe vývoja je venovaná tvorbe presnej matematickej špecifikácie, ktorá by zodpovedala vytváranému produktu. Výsledný produkt, systém a dokumentáciu, môžu z takto vytvorenej špecifikácie vytvárať automatizované generátory.

Špirálový model

Tento model graficky znázorňuje špirála, kde každý jeden cyklus predstavuje jednu fázu vývoja. Základ tohoto modelu je prebratý z vodopádového procesného modelu. Cykly špirály postupne smerom od stredu ku okrajom špirály predstavujú jednotlivé fázy procesu a zameriavajú sa postupne na analýzu uskutočniteľnosti projektu, špecifikáciu systémových požiadaviek, systémový návrh atď. V modeli nie sú presne určené jednotlivé fázy, preto aj štruktúrovanie projektu záleží na rozhodnutí manažmentu.



Obr. 7. Schéma špirálového modelu [1][3][4]

Každý cyklus špirály je rozdelený na štyri časti (pozri obrázok 7):

1. *Určenie cieľov* Určenie špecifických cieľov pre danú fázu projektu. Identifikujú sa ohraničenia procesu a produktu a vypracuje sa detailný riadiaci plán. Určujú sa možné riziká a vypracovávajú sa náhradné stratégie podľa identifikovaných rizík.
2. *Stanovenie rizík a ich eliminácia* Pre každé identifikované riziko sa vypracováva detailná analýza. Vykonajú sa potrebné kroky na elimináciu týchto rizík.
3. *Vývoj produktu a validácia* S ohľadom na dané riziká sa vyberie jeden model pre vývoj systému(pri veľkých bezpečnostných rizikách to môže byť napr. formálna transformácia). Výber modelu je pre jednotlivé fázy vývoja alebo cykly špirály nezávislý.
4. *Plánovanie* Po preskúmaní súčasného stavu projektu sa rozhodne či je potrebný ďalší cyklus špirály. Ak sa bude v procese pokračovať potom sa v tejto etape naplánuj celá ďalšia fáza vývoja.

Prínosom špirálového modelu bolo spojenie výhod iteratívneho spôsobu vývoja produktu s výhodami vývoja v navzájom oddelených fázach. Model poskytuje dobrú viditeľnosť vývoja projektu a zjednodušuje aj tvorbu dokumentácie k produktu, podobne ako je tomu pri vodopádovom modeli. Zároveň však umožňuje viackrát sa vrátiť k jednotlivým fázam tvorby, bez čoho by nebolo možné tento model použiť pre

tvorbu nových alebo príliš veľkých systémov, kde je veľké riziko zlého pochopenia požiadaviek zákazníka. Najväčšou výhodou špirálového modelu je priame zahrnutie rizík (identifikácie, analýzy a eliminácie rizík) do modelu tvorby softvéru. Riziko je miera neurčitosti výsledku a zvyčajne je spôsobené nedostatkom presných informácií. Riešením rizika je činnosť, ktorá zabezpečí dostatok informácií, ktoré eliminujú neurčitosť a znížia pravdepodobnosť nastatia rizika pod mieru zanedbateľnosti. Model uľahčuje manažment rizík presne definovanou šablónou pre každý cyklus špirály. Šablóna sa vypĺňa postupne počas celého cyklu a obsahuje nasledujúcich osem bodov:

1. *Úlohy* Identifikácia cieľov, ktoré sú pre daný cyklus špirály najdôležitejšie.
2. *Ohraničenia* Faktory obmedzujúce možné riešenia alebo postupy pri riešení.
3. *Alternatívy* Rôzne spôsoby plnenia úloh.
4. *Riziká* Možné riziká spájajúce sa s identifikovanými alternatívami.
5. *Riešenie rizík* Postupy eliminácie rizík.
6. *Výsledky* Závery vyplývajúce z riešenia rizík.
7. *Plány* Zostavenie ďalšieho cyklu špirály.
8. *Záväzky* Rozhodnutia manažmentu o tom ako bude projekt pokračovať.

V rámci špirálového modelu môžeme v jednotlivých fázach vývoja použiť všetky ostatné modely softvérového procesu podľa dominantných rizík pre daný cyklus. Explicitné uvažovanie rizík v softvérovom procese odlišuje tento model od ostatných.

Zhrnutie vlastností klasických modelov

Klasické modely tvorby softvérových procesov, ktoré sme v krátkosti prezentovali, predstavujú v zásade vždy trochu odlišný prístup k tvorbe softvéru. Nie sú to teda len vývojové stupne vyvíjajúceho sa modelu tvorby softvérových produktov, ale sú to samostatné modely, z ktorých každý prináša so sebou určité výhody aj nevýhody. To za akých podmienok sa oplatí použiť niektorý model stručne popíšeme v tejto časti.

Počas vývoja produktu je potrebné zabezpečiť efektívne riadenie projektu. Aby sa to dalo zabezpečiť, musí mať projekt niektoré dôležité vlastnosti. Z týchto vlastností môžu na výber modelu tvorby softvéru vplývať hlavne nasledujúce vlastnosti:

1. Zrozumiteľnosť: do akej miery je proces explicitne definovaný daným modelom a koľko snahy si vyžaduje jeho pochopenie?
2. Viditeľnosť: nakoľko je postup prác v procese vývoja podľa daného modelu externe viditeľný?
3. Podpora CASE nástrojov: podporujú spôsob vývoja podľa daného modelu niektoré CASE nástroje?

4. Spoľahlivosť: je proces navrhnutý daným modelom tak, aby sa chyby v procese vývoja neprejavili chybami vo výslednom produkte?
5. Robustnosť: umožňuje proces tvorby podľa daného modelu pokračovať vo vývoji aj po vzniku neočakávaných problémov?
6. Udržiavateľnosť: možno proces tvorby v danom modeli prispôsobiť zmeneným organizačným požiadavkám?
7. Rýchlosť vývoja : ako rýchlo možno vytvoriť produkt podľa daného modelu?

Okrem vlastností, ktoré sa vyžadujú od procesu tvorby softvéru, môžu na výber modelu vývoja vplývať aj vlastnosti, ktoré sa vyžadujú od vytváraného produktu. Môžu to byť nasledujúce vlastnosti:

1. Udržiavateľnosť: bude možné softvér dodatočne prispôsobovať novým požiadavkám zákazníka?
2. Spoľahlivosť, bezpečnosť: vytvorený softvér nesmie spôsobiť žiadne fyzické či ekonomické škody v prípade jeho poruchy alebo výpadku.
3. Výkonnosť: softvér nesmie plytvať prostriedkami systému.
4. Použiteľnosť: softvér musí mať vhodné používateľské rozhranie a dostupnú dokumentáciu.

Niekoľkými slovami môžeme popísať jednotlivé klasické modeli nasledovne:

1. Vodopádový model: priveľmi abstraktný, neuvažuje iterácie v procese tvorby, dobre zdokumentovateľný, dobre viditeľný proces vývoja
2. Evolučný model: jednoduchý, vývoj prebieha v cykloch, umožňuje dobré pochopenie zákazníkových požiadaviek, problém vzniká pri štruktúrovaní vytváraného produktu a pri tvorbe dokumentácie
3. Formálna transformácia: zabezpečuje bezchybovú transformáciu formálnej špecifikácie na výsledný produkt, poskytuje veľký priestor pre možné zautomatizovanie tvorby softvéru, problém je vytvoriť formálnu špecifikáciu
4. Špirálový model: pravdepodobne najuniverzálnejší model - môže mať ľubovoľný počet cyklov, každý cyklus môže použiť iný model tvorby softvéru podľa aktuálnych priorít, presne definovaná šablóna pre každý cyklus zabezpečuje dobrú viditeľnosť a ľahkú zdokumentovateľnosť procesu tvorby softvéru, priamo uvažuje možné riziká

Použitá literatúra

[1] Jalote, P.:An Integral Approach to Software Engineering, Springer-Verlag New York, 1997

[2] Bieliková, M.: Manažment v softvérovom inžinierstve, 1999

[3] S.Badr: A Model and Algorithms for Software Evolution Control System, doctoral dissertation, Naval Postgraduate School, Monterey, Calif.,1993

Sommerville, I.: Software Engineering, Addison-Wesley

Formálne pozadie evolučného vývoja softvéru

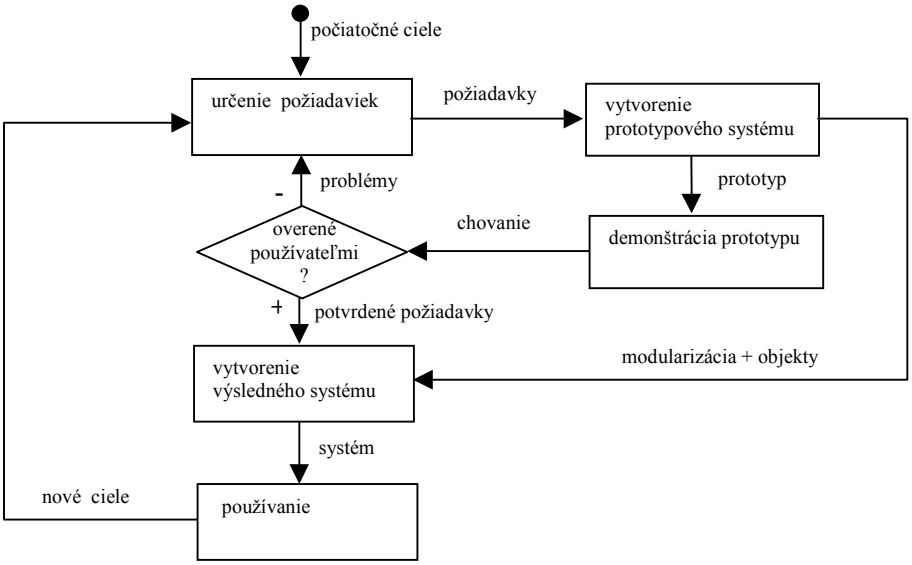
Branislav Lehotský

Abstrakt. Evolučný softvérový proces patrí už ku klasickým postupom používaných v softvérovom inžinierstve. Tento štýl tvorby so sebou však prináša i základné problémy (horšia viditeľnosť postupu, častá zlá štruktúrovanosť systému, vyžadovanie špeciálnych schopností). Nevýhody môžu byť eliminované vhodnými vývojovými nástrojmi a tie musia zase pracovať na základe vhodného formalizmu. Článok prezentuje formálny model, podporujúci evolučný vývoj softvéru.

Flexibilita prototypovania

Podrobne sa problematikou evolučného softvérového procesu zaoberá I.Sommerville [3]. Okrem iného rozlišuje a popisuje dva typy evolučného vývoja: objavné programovanie (exploratory programming) a prototypovanie na zahodenie (throw-away prototyping). Objavné programovanie sa použije, keď je obtiažne alebo nemožné stanoviť podrobnú špecifikáciu systému.

J.A.Goguen a Luqi popisujú vývoj založený na prototypovaní takto [1]:



Obr. 8 Schéma iteratívneho prototypového životného cyklu

Používatelia a návrhári spolu pracujú na definovaní požiadaviek. Návrhári vytvoria prototyp a ten je potom spätne demonštrovaný, aby sa overilo či jeho chovanie zodpovedá očakávanému. To umožní identifikovať problémy a môže viesť k predefinovaniu požiadaviek. Tento proces sa opakuje, kým prototyp „uspeje“ a podchycuje všetky požadované ťažiskové aspekty. Návrhári potom použijú takto overené a potvrdené požiadavky ako základ pre výslednú produkciu. Týmto spôsobom môžu byť softvérové systémy dodávané postupne a analýza požiadaviek môže pokračovať počas celého vývoja. Používatelia tak majú so softvérom veľmi rýchlo nové skúsenosti, môžu formulovať ďalšie ciele a usmerňovať jeho nasledujúce iterácie.

Formálne metódy pre evolučný vývoj

Pri takom spôsobe tvorby softvéru, kde sú nevyhnutné časté následné zmeny a početné iterácie, môžu byť veľmi užitočné rôzne formálne metódy na podporu a podchytenie týchto procesov. Vývojové nástroje založené na špecifických formálnych modeloch pomôžu udržiavať integritu projektu mnohými spôsobmi. Konkrétne pri plánovaní úloh, sledovaní dodržiavania termínov, sledovaní dôvodov pre vznik objektov a zmien a pri udržiavaní závislostných relácií medzi jednotlivými

verziami, variáciami a dekompozíciou komponentov. Na podporu týchto mechanizmov bol navrhnutý i **hypergrafový model** [1],[2].

Hypergrafy rozširujú a zovšeobecňujú známe orientované grafy pridaním *hyperhrán*, ktoré môžu mať viaceré vstupné vrcholy a viaceré výstupné vrcholy. Nasleduje zjednodušený matematický aparát s opisom:

Def.1: Hypergrafom nazývame štvoricu (N,E,I,O), kde:
N je množina *vrcholov*,
E je množina *hyperhrán* (zjednodušene hrán)
I: E → 2^N je fukcia určujúca množinu *vstupov* pre každú hyperhranu a
O: E → 2^N je fukcia určujúca množinu *výstupov* pre každú hyperhranu

V hypergrafovom softvérovom evolučnom dátovom modeli reprezentuje každý vrchol nejaký softvérový komponent, čo je nemeniteľná verzia softvérového objektu. Softvérové objekty môžu byť pritom časti a moduly samotného programu, ale aj najrozličnejšieho iného druhu, napríklad: prehľady problémov, požiadavky na zmenu, reakcie na demonštrácie prototypov, požiadavky, špecifikácie, manuály, dokumenty návrhu, testovacie dáta, atď. Hyperhrany (hrany) vyjadrujú závislosti množstva rôznych verzií softvérových objektov v systéme a reprezentujú tak evolučné kroky (vývojové aktivity, ktoré vytvárajú výstupné objekty O(e); e∈E). Tieto závislosti reprezentujú podstatu histórie odvodzovania i plány na budúcu evolúciu.

Def.2: Evolučný hypergraf je hypergraf H = (N,E,I,O) spolu s funkciami L_N: N→C a L_E: E→A, spĺňajúcich nasledujúce pravidlá:

1. N a E sú disjunktné podmnožiny množiny U, ktorej prvky sa nazývajú *jedinečné identifikátory*
2. Ak O(e) ∩ O(e') ≠ Ø, potom e = e'
3. H je acyklický a A={s,d}x A' (t.j.: každý prvok A má tvar [s,a'], alebo [d,a'], pričom a' ϵ A')

   Hrana označená „s“ sa nazýva krok (step), hrana označená „d“ sa nazýva dekompozícia.

Prvky množiny N sú identifikátormi softvérových komponentov, prvky množiny E sú identifikátormi evolučných krokov a I a O určujú vstupy a výstupy pre každý evolučný krok. Funkcia L_N označuje každý vrchol atribútmi z množiny C a zodpovedajúcou značkou verzie softvérového objektu. Funkcia L_E označuje každú hranu atribútom kroku z množiny A a aktuálnym stavom kroku.

Prvé pravidlo požaduje, aby identifikátory vrcholov a hrán boli odlišné, jedinečné. Druhé pravidlo stanovuje, aby výstupné množiny dvoch rôznych evolučných krokov boli disjunktné (teda každý evolučný krok je jednoznačne identifikovateľný udaním ľubovoľného komponentu ktorý produkuje). Tretie pravidlo stanovuje, že proces softvérovej evolúcie nás nikdy neprivedie späť ku

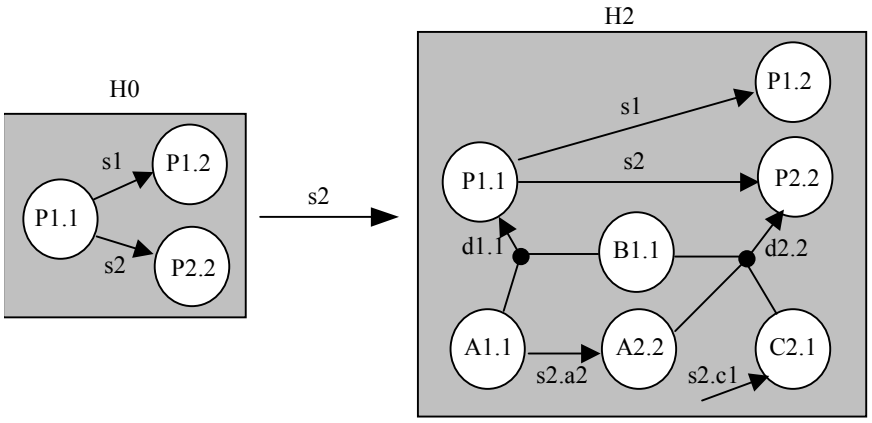
komponentom, ktoré už boli vytvorené v minulosti a tiež, že jeden identifikátor nemožno použiť opakovane.

Posledné definície rozšíria model o možnosť, aby niektoré evolučné kroky mohli pozostávať z iných (na nižšej úrovni), teda aby ich bolo možné „rozvinúť“. Na zrealizovanie tejto myšlienky sa zavádza hierarchická štruktúra hyperhrán v hypergrafe:

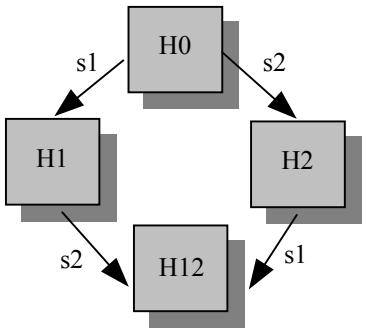
Def.3: *Hierarchický hypergraf* je acyklický graf, ktorého vrcholy tvoria hypergrafy, pričom má len jeden list a jeden koreň; každá jeho hrana zodpovedá jednému rozvinutiu jednoduchej hyperhrany v hypergrafe v jej počiatku, výsledkom ktorého je hypergraf v jej konci; a výsledok kompozície rozvinutí pozdĺž dvoch ciest medzi tými istými dvoma uzlami je rovnaký.

Hierarchický evolučný hypergraf je hierarchcký hypergraf, ktorého vrcholy tvoria evolučné hypergrafy a ktorého hrany zodpovedajú kroku, ktorý bol rozvinutý.

Hypergraf v korennom uzle je najabstraktnejším pohľadom (najvyššej úrovne) evolučnej histórie a štruktúry systému, hypergraf v liste predstavuje úplne rozvinutý pohľad resp. formu. Vrcholy koreného hypergrafu sú rozličné verzie celého softvérového produktu. Vrcholy listového hypergrafu obsahujú verzie atomických (ďalej nedeliteľných) softvérových objektov tvoriacich softérový produkt a všetky kroky v listovom hypergrafe sú tiež atomické.



Obr. 9 Jednoduchý ilustračný príklad



Obr. 10 Jednoduchý ilustračný príklad

Ľavá časť na Obr. 9 obsahuje tri verzie softvérového systému (označené P1.1, P1.2, P.2.2). Krok s1 vytvára a odvádza novú verziu P1.2 z P1.1, krok s2 vytvára inú variantu P2.2 z P1.1. To je príklad jednoduchého evolučného hypergrafu. Pravá časť na Obr. 9 zobrazuje príklad možného rozvinutia hrany s2.

Tieto dva popisy histórie a štruktúry systému zodpovedajú dvom hypergrafom označeným H0 a H2 na Obr. 10. Hrana z H0 do H2 zodpovedá rozvinutiu evolučného kroku s2. Ak rozvinieme aj krok s1 zodpovedajúci hrane z H0 do H1, potom musíme mať aj štvrtý hypergraf H12, obsahujúci obidve rozvinutia hrán. Kompletný hierarchický evolučný hypergraf je na Obr. 10, kde H0 je koreň a H12 uzol

Cieľom tejto časti bolo načrtnúť matematické pozadie, na základe ktorého môžu vývojové nástroje (pre počítačom-podporovaný návrh) podporovať evolučnú tvorbu softvéru a jej riadenie, môžu v softvérovom projekte evidovať a riadiť jednak aktivity (zmeny, zásahy) ale aj výsledky, ktoré sú produktmi týchto aktivít. Uvedený hypergrafový model je tak schopný reprezentovať evolučnú históriu i budúce plány vývoja softvérového projektu.

Použitá literatúra

[4] Luqui and J. A. Goguen: „Formal Methods: Promises and Problems“. IEEE Software, January 1997, str. 73-85.

[5] S. Badr: A Model and Algorithms for Software Evolution Control System. Doctoral dissertation, Naval Postgraduate School, Monterey, Calif., 1993.

[6] I. Sommerville: Software Engineering. Addison-Wesley.

Softvérový proces pri tvorbe Web aplikácií

Ján Glončák

Abstrakt. Inžinierstvo procesu vývoja softvéru umožňuje dosiahnuť omnoho lepšie výsledky pri vývoji softvéru. Tým, že definuje pravidlá, vnáša do vývoja softvéru značnú mieru formálnych postupov, ktoré je nutné dodržiavať. Formálne postupy zdanlivo zväčšujú réžiu vývoja, ale sú nevyhnutné v prípade veľkých softvérových projektov. Rozsiahle web aplikácie možno považovať za veľké projekty, pretože pri ich vývoji úzko spolupracujú programátori, databázoví špecialisti, dizajnéri a ostatní členovia vývojového tímu. Preto dodržiavanie inžinierstva procesu vývoja softvéru a správna identifikácia etáp životného cyklu napomáha k zvýšeniu kvality vývoja softvéru ako aj samotného softvéru.

Životný cyklus vývoja web aplikácie

Životný cyklus tvorby web aplikácie možno rozdeliť na nasledovné etapy: konzultácia, koncept, výrobný proces, prezentácia web aplikácie, zverejnenie web aplikácie, ukončenie projektu web aplikácie,

– Konzultácia

Prvý krok je stretnutie predstaviteľa vývojového tímu a klienta, kde prediskutujú kľúčové body projektu. Klient opíše o čo má záujem a čo chce web aplikáciou prezentovať. Vývojár oboznámi klienta s možnosťami web aplikácií, ich typmi a cenovými reláciami.

– Koncept (plán)

Po konzultácií a úvodnej špecifikácií požiadaviek sa vytvorí koncepčný návrh stránok, kde je naznačená úvodná stránka, navigácia, rozmiestnenie stránok nižších úrovní.

Etapa má dve časti:

- vývoj obsahu – definovanie štruktúry web aplikácie

- predbežná funkcionálna špecifikácia – definovanie základných funkcií web aplikácie, môže to byť napr. vyhľadávanie údajov v databáze a ich zverejňovanie na Internete, alebo elektronický obchod, atď...

Na tejto úrovni sa stanovuje cenová relácia a predbežný pracovný plán. Je to etapa vývoja web aplikácie, ktorá predstavuje pre klienta poslednú možnosť zásadnej zmeny projektu, preto je veľmi dôležité nepodceniť jej význam a venovať jej dostatočné množstvo času.

– **Výrobný proces**

Keď je koncept schválený, zmeny a pripomienky boli akceptované môže začať výroba web stránky. Úloha klienta v tejto časti je poskytovať podklady na tvorbu stránok ako sú texty, poprípade obrázky.

Z pohľadu vývojárov táto etapa obsahuje:

- vývoj grafického návrhu – predstavuje vytvorenie grafického návrhu web aplikácie
- vývoj architektúry systému – definovanie štruktúry adresárov jednotlivých stránok, návrh údajových zdrojov (najčastejšie návrh štruktúry databáz, tabuliek, ich atribútov a relácií)
- vývoj používateľského navigačného rozhrania – realizácia navigácie web aplikácie, predstavuje ju rozmiestnenie navigačných tlačidiel, návrh spôsobu jednoduchej identifikácie jednotlivých častí web aplikácie
- vývoj funkcionálnej špecifikácie – vytváranie funkcionálnych častí web aplikácie, ktoré sú obvykle členené do modulov zaoberajúcich sa navzájom nezávislými funkciami ako napr. vyhľadávanie údajov a ich distribúcia
- implementácia web aplikácie - tvorba web dokumentov, programovanie skriptov, apletov a pod.

V priebehu vývoja je dobrým zvykom prezentovať klientovi čiastočné výsledky. Je to istý spôsob prevencie pred „veľkými záverečnými zmenami“.

– **Prezentácia web aplikácie**

Po ukončení tvorby web aplikácie ešte pred zverejnením stránky sa musí poskytnúť klientovi možnosť záverečnej revízie. Za ňou nasleduje úprava aplikácie podľa pripomienok a potvrdenie vykonania zmien.

– **Zverejnenie web aplikácie**

Umiestenie stránky do siete Internet prípadne aj registrácia vo vyhľadávačoch.

– **Ukončenie projektu vývoja web aplikácie**

Zverejnením web aplikácie sa nekončí jej životný cyklus. Je veľmi dôležité identifikovať ukončenie projektu vývoja web aplikácie ako etapu životného cyklu a vyhradiť si na ňu určitý čas, pretože jej správna realizácia pomáha pri údržbe softvéru a znovupoužívaniu vytvoreného softvéru pri vývoji podobných aplikácií.

Základným cieľom tejto etapy je vytvoriť správu o ukončení projektu. Nie je to záverečná dokumentácia a slúži hlavne pre tím vývojárov na analýzu práce. Mala by poskytovať možnosť spätného pohľadu na ukončený projekt alebo na nejakú jeho etapu. Spravidla obsahuje zhrnutie toho čo sa vykonalo, čo bolo jednoduché a čo bolo komplikované, návrhy na zlepšenie riešenia týchto procesov, zoznam nepredvídaných udalostí.

pozn. V niektorých firmách sa zvyknú vykonávať aj kritické posudky seberovných vývojárov (PEER REVIEWING) [2], kde členovia tímu, ktorí pracujú v tej istej oblasti hodnotia poprípade kritizujú prácu a pracovné postupy svojich kolegov. Kritické posudky akosti vytvorenej aplikácie (môžu byť aj za každou etapou vývoja) sú vo vývojovom tíme veľmi neobľúbené, ale napomáhajú k zlepšeniu kvality vyvíjaného softvéru. Spravidla dokážu odstrániť chyby, na ktoré by autor neprišiel. Pri kvalitnej dokumentácií umožňujú analýzu výkonnosti a kvality programovania jednotlivých členov tímu.

– **Údržba web aplikácie**

Web aplikácia môže byť rozširovaná a modifikovaná aj v prípade, že už je zverejnená. Spôsob údržby stránky musí byť definovaný a presne dohodnutý s vývojármi už v etape plánovania.

Metódy zefektívnenia procesu vývoja web projektov

Aj po správnej identifikácií etáp životného cyklu sa dá ešte proces vývoja web aplikácií vylepšovať. Naznačím pár zásad týkajúcich sa manažmentu projektu, ktoré môžu zvýšiť efektivitu v oblastiach plánovania, riadenia zmien, stanovovania priorít projektov a pri vývojovom tíme [2].

– **Plánovanie**

Je veľmi dôležité stanoviť plán, ktorý je dostatočne podrobný a presne popisuje celý rozsah projektu a tým poskytuje možnosť kontroly priebehu vývoja. Manažér projektu musí zabezpečiť aj stanovenie štandardných dokumentov a šablón používaných pri procesoch vývoja aplikácie. Pri plánovaní sa musia presne definovať aj funkcie a zodpovednosti jednotlivých členov vývojového tímu.

– **Riadenie zmien**

Aj napriek relatívnej jednoduchosti vykonávania zmien je aj pri web dizajne na efektívne spracovanie zmien nevyhnutná disciplína a dodržiavanie pravidiel. Treba rozlišovať tieto základné typy zmien:

- požiadavky na nový projekt
- požiadavky na zmeny práve vyvíjaného softvéru
- požiadavky na riešenie problémov súvisiacich s „bežiacou“ aplikáciou
- požiadavky na rozšírenia súčasného systému
- obsahové zmeny v súčasnom systéme

Veľmi dôležité je definovanie presných pravidiel na komunikáciu medzi klientom a vývojovým tímom.

Je vhodné používať štandardné typy dokumentov a formulárov (tie by mali byť navrhnuté už v procese plánovania) a všetky zmeny evidovať. Každú zmenu treba zvážiť a pred jej vykonaním ju treba oznámiť klientovi. Členovia vývojového tímu sa musia vyvarovať samovoľných zmien a vylepšovaní web aplikácie.

– **Stanovenie priorít projektov**

Väčšina spoločností orientujúcich sa na web dizajn musí riešiť viacero požiadaviek projektov naraz a preto je nevyhnutné stanoviť štandardný model na určenie priority projektov. Pretože množstvo zdrojov je obmedzené treba identifikovať najvhodnejší projekt na riešenie.

Základné faktory na určovanie priority ponúkaných projektov sú:

- nevyhnutnosť odkladu
- úroveň technologického rizika
- miera, ktorou projekt ovplyvní zdroje potrebné na súčasný vývoj
- obchodná hodnota
- stupeň komplexnosti používateľského rozhrania

– **Vývojový tím**

Pre každý projekt rozsiahlej web aplikácie je vytvorený tím, ktorý pozostáva zo špecialistov z jednotlivých oblastí. Typický projektový tím musí mať vedúceho projektu, technického vedúceho, vedúceho dizajnu, predstaviteľa obchodnej časti, softvérového vývojára, databázového experta a v prípade veľmi veľkých projektov je vhodné mať v tíme aj personalistu.

Pri snahe o zvýšenie efektivity práce tímu je vhodné zamerať sa na zlepšenie procesu v týchto činnostiach:

- spracovanie veľkého množstva novej práce

- vyvinúť základ pre bežné procesy na pomoc novým členom stať sa platnými členmi tímu
- udržiavať vysokú kvalitu práce aj pri zvyšovaní produktivity
- efektívne výmena informácií a vedomostí medzi členmi tímu

Použitá literatúra:

[1] Karl Wiegers: Software Process Improvement in Web Time. IEEE Software, July/August 1999, str 78-86.

[2] William Lewis: Web Development Process. January 2000. http://www.wmlewis.com

Extrémne programovanie

Marián Varga

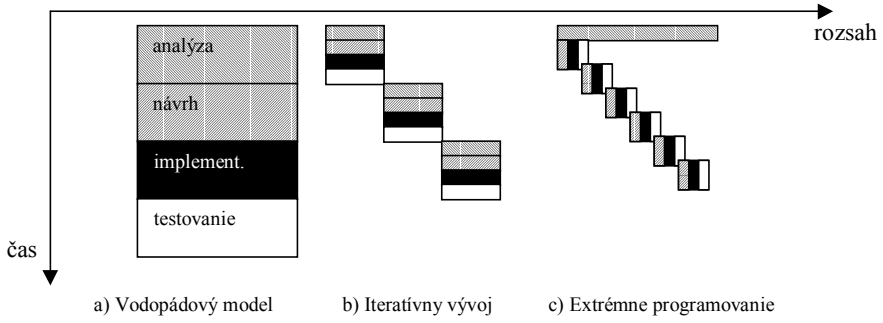
Abstrakt. Extrémne programovanie je model procesu vývoja softvéru, ktorý skracuje vývojový cyklus na minimum. Spoliehajúc sa na flexibilitu programu a neustály dialóg so zákazníkom pridávajú programátori iba tie funkcie a prvky návrhu, ktoré sú potrebné podľa momentálneho stavu zákazníckych požiadaviek. Pri meniacich sa požiadavkách je takto možné vytvoriť stabilný a spoľahlivý produkt za kontrolovateľný čas.

Úvod

Ako reakcia na zlyhania triviálneho procesu vývoja „Vytvor a oprav“ (Build and Fix) [1] vznikol najprv vodopádový model procesu vývoja, ktorý rozčlenil proces na etapy (Obr. 11a), pričom pred implementáciou musia najprv prebehnúť fázy analýzy a návrhu. Cieľom je urobiť najvplyvnejšie rozhodnutia čo najskôr, pretože čím neskôr je potrebné urobiť zmenu, tým je to drahšie [1].

Postupne sa však ukázalo, že používateľ prakticky nikdy nie je schopný úplne a správne formulovať svoje požiadavky vo fáze analýzy. Navyše sa počas dlhého vývojového cyklu často požiadavky menia. Na zlepšenie dialógu s používateľom vznikli procesy používajúce iteráciu kratších vývojových cyklov (Obr. 11b).

Súčasne sa však začali v programovaní rozvíjať nové techniky (väčšina z nich súvisí s objektovo orientovaným programovaním, OOP), ktoré významne uľahčili zmeny v programe a zmiernili ich dopad. Možnosť písať flexibilné programy sa stala základom Extrémneho programovania (Extreme Programming, XP) – procesu, ktorý extrémne skracuje vývojové cykly, takže namiesto investovania do ďalekej a neistej budúcnosti sa pri ňom vykonávajú činnosti plánovania, analýzy, návrhu, implementácie a testovania vlastne súčasne a podľa momentálnych potrieb (Obr. 11c).



Obr. 11 Porovnanie extrémneho programovania z hľadiska rozloženia činností v čase s tradičným vodopádovým modelom a všeobecným iteratívnym vývojom

Na začiatku procesu XP treba v analýze určiť, čo by systém mohol robiť, a čo by mal robiť prvé [1]. Hneď potom nasleduje jednoduchý návrh a implementácia vybranej malej skupinky požiadaviek. Cieľom je za čo najkratší čas (približne mesiac) vytvoriť už fungujúci program, hoci s úzkou funkcionalitou. Požiadavky používateľov sa špecifikujú pomocou tzv. „príbehov“ (Stories), ktoré sú vlastne prípadmi použitia v rozsahu malej kartičky, vysvetľujúce funkciu programu pomocou príkladov. Zo všetkých „príbehov“ si potom používateľ vyberie malú skupinku, ktorá sa ako prvá implementuje.

Počas práce vývojového tímu na zvolených funkciách používateľ špecifikuje (akceptačné) testy systému. Vývojový tím rozdelí vybrané „príbehy“ na jednotlivé úlohy, ktoré treba pri ich implementácii vykonať a pridelí ich dvojiciam programátorov. Implementuje sa v poradí významu (ceny) jednotlivých „príbehov“ z hľadiska používateľa. Ešte pred implementáciou však programátori napíšu automatické testy súčiastok, ktoré idú implementovať. Testy slúžia ako indikátor toho, že súčiastka je hotová. Okrem toho sa testy v systéme navždy uchovávajú a ich spúšťaním sa neustále overuje, či zmeny v programe nespôsobili chybné fungovanie niektorej súčiastky.

Po dokončení všetkých „príbehov“ zvolených pre aktuálnu iteráciu vývoja sa overuje splnenie používateľských požiadaviek pomocou testov systému. Kým testovaním súčiastok sa buduje dôvera programátorov, testovanie systému získava dôveru zákazníka vo funkčnosť celého programu. Používateľ získava z procesu vývoja veľmi rýchlu odozvu, na ktorú môže reagovať zmenou požiadaviek alebo prehodnotením ich priority.

Princípy Extrémneho programovania

Extrémne programovanie je „odľahčený“ proces, ktorý sa sústreďuje na to najpodstatnejšie, čo každý proces musí obsahovať – **programovanie**

(implementáciu). Aby sa implementovali správne veci, treba aj **počúvať** používateľove požiadavky. Aby sme vedeli, kedy program funguje, musíme ho **testovať**. Keďže sa v XP vždy počíta s veľa budúcimi zmenami v programe, tento musí byť nutne jasný a jednoduchý. V takom poriadku ho treba udržovať **preštruktúrovaním** častí zdrojového kódu. Dá sa teda povedať, že podstatu XP charakterizuje skratka „LCTR“ (počúvanie – Listening, programovanie – Coding, testovanie – Testing a preštruktúrovanie – Refactoring). [3]

Aby sa zaručilo fungovanie „LCTR“, extrémne programovanie požaduje dodržiavanie týchto 12 princípov [3]: Testovanie súčiastok, Programovanie v pároch, Nemilosrdne preštruktúrovať, Nebude to treba, Testovanie systému, Spolupracujúci zákazník, Neustála integrácia, Plánovacia hra, Systémová metafora, Časté zostavovanie, Kolektívne vlastníctvo kódu a Programovacie konvencie. Bližšie sa im venujú nasledujúce časti.

- **Testovanie súčiastok (Unit Tests)**

Ide o hlavnú techniku Extrémneho programovania. Vďaka nej je možné systém zložený z jednotlivých súčiastok (tried) udržiavať už od začiatku vo fungujúcom stave, hoci sa doňho stále pridávajú funkcie a zdrojový kód sa „preštruktúruje“, aby bol program čo najjednoduchší.

Súčiastky musia prejsť všetkými testami na 100%, inak nemožno pokračovať skôr, ako sa chyby opravia. Hneď, keď sa zistí, že testom prešli aj chybné súčiastky, treba testy rozšíriť tak, aby zachytili tieto chyby, ako aj všetky potenciálne chyby, na ktoré dovtedy nemyslelo.

Testovanie súčiastok sa vykonáva pomocou programov, ktoré generujú testovacie vstupy a kontrolujú výstupy testovanej súčiastky. Vďaka tomu je možné ich neustále spúšťať na overenie, že v systéme pri zmenách nevznikli chyby. To je však možné, iba ak sú testy navrhnuté tak, aby nebežali dlhšie ako 10 minút, inak by neustále testovanie významne brzdilo vývoj.

- **Programovanie v pároch (Pair Programming)**

Podľa XP programujú vždy dvaja programátori pri jednom počítači (klávesnici, monitore, myši). Tým je zabezpečená vzájomná kontrola, odovzdávanie skúseností a motivácia.

Tvorcovia softvéru trávia 30% času samostatnou prácou, 50% času prácou vo dvojici a 20% prácou s viac ľuďmi [3]. Iba pri samostatnej práci sa väčšinou pracovníci venujú svojej práci – sústredenie (flow). Takéto sústredenie však môže vzniknúť aj pri práci vo dvojici a práve o to sa usiluje XP.

- **Nemilosrdne preštruktúrovať (Refactor Mercilessly)**

Súvisí s princípom „všetko povedať iba raz“: Navzájom podobné úseky programu (metódy alebo objekty v OOP) treba nahradiť jedným spoločným. Nepriaznivé vplyvy týchto zmien možno zamedziť rozsiahlym a opakovaným testovaním súčiastok a testovaním systému.

Systém sa nesmie stať rukojemníkom žiadnej zastaralej, skrytej, či zle pochopenej myšlienky z minulého vývoja. Naopak, pri XP je program stále čerstvý, produktívny a je radosť na ňom pracovať [3].

Keďže pri pridávaní funkcií systému „robíme to najjednoduchšie, čo by mohlo fungovať“, neskôr sa môže ukázať, že štruktúra programu po takejto zmene by sa mala upraviť. Tieto úpravy, ktoré spôsobujú, že návrh systému lepšie zodpovedá tomu, na čo slúži, sú vítané, pretože sa vykonávajú práve vtedy, keď sa skutočne preukáže, že sú potrebné. Nie sú teda založené iba na neistej (predčasnej) predstave, že niekedy v budúcnosti budú pravdepodobne potrebné [3].

Súvislosť s potrebou krátkeho vývojového cyklu: Čím dlhšie má program konkrétnu podobu zdrojového kódu a naozaj spĺňa stanovené požiadavky, tým lepšie vieme zlepšovať a rozširovať jeho vnútornú štruktúru.

Čo ak pri preštruktúrovaní zlúčime iba náhodne sa podobajúce, no v skutočnosti nesúvisiace, časti programu, čím návrh namiesto zlepšenia zhoršíme? Fakt je, že neexistujú objektívne kritériá na určenie, či sa časti programu podobajú „iba náhodou“. Pragmatický prístup velí „počúvať tomu, čo hovorí program“, namiesto arogantného domnievania sa, že my vieme viac [3]. Ak by sa predsa ukázalo, že spojenie častí nebolo šťastné (v ďalšom vývoji začnú pribúdať odlišnosti), ako „poistka“ môže slúžiť, ak sa zachovajú odlišné pomenovania spájaných častí (napr. metód), pričom implementácia jednej z nich sa nahradí presmerovaním na druhú. Potom v prípade potreby zrušenia tohto spojenia nie je potrebné vracať späť všetky odkazy (volania) smerujúce na tieto časti. Tým sa dosiahne, že prinajmenšom dočasne sa program zjednoduší a súčasne bude pružný voči zmene [3].

Vo väčšine prípadov by sa pri preštruktúrovaní mala meniť najmä implementácia a nie rozhrania. Zmeny rozhraní totiž znamenajú, že súčiastky majú navonok akoby inú funkciu, a teda pre ne treba pripraviť aj zodpovedajúcu novú súpravu testov súčiastok.

- **Nebude to treba (You Aren’t Gonna Need It, YAGNI)**

Programátori majú sklon včleňovať do návrhu programu vlastnosti, ktoré nie sú potrebné pre práve vyvíjanú verziu programu, ale o ktorých predpokladajú, že budú potrebné pri vývoji neskôr.

Argumenty v prospech takéhoto konania:

A1. niekedy v budúcnosti sa tým môže urýchliť vývoj

A2. programátor je presvedčený, že práve teraz vie, ako to urobiť, alebo pracuje v blízkosti na to vhodného miesta programu

A3. v budúcnosti sa to bude robiť ťažšie

Protiargumenty:

B1. spomalí sa práca na aktuálnej úlohe

B2. zvyšuje sa riziko oneskorenia iterácie i celého projektu

B3. nakoniec to môže byť zbytočné

B4. môže to trvať dlhšie, ako sa domnievame

B5. kým to skutočne nebude potrebné, nebudeme vedieť, ako to správne urobiť

B6. od momentu implementácie bude potrebná údržba

Ad A2: Riziku, že ani sám programátor nebude schopný v budúcnosti svojmu programu rozumieť a pridať doňho potrebnú vlastnosť, je lepšie predísť (udržiavaním jednoduchého návrhu).

Dôležité je uvedomiť si, že máme pokročilé techniky na minimalizáciu efektu zanesenia chýb následkom zmien (OOP, XP), ale nemáme dobré techniky na predpovedanie budúcnosti.

Zaujímavý je pohľad na Problém roku 2000 z hľadiska princípu YAGNI [3]: Pri systémoch, ktoré neukladali do dátumov storočie, sa vo veľkom počte prípadov dosiahla úspora, pretože veľa z týchto systémov bolo (z iných dôvodov ako samotný rok 2000) vyradených z prevádzky pred rokom 2000, takže príprava na rok 2000 by bola v nich zbytočne. Keď však v nich bol 2-ciferný rok zavedený úmyselne namiesto prirodzeného 4-ciferného, potom išlo o porušenie YAGNI, tzv. predčasnú optimalizáciu (budúcnosť ukázala, že táto optimalizácia nebola správna a lepšie by bolo bývalo vôbec ju nerobiť).

Iný názov princípu YAGNI: Práve včas (Just in Time, JIT), podobnosť názvu s kompilátormi JIT nie je náhodná: tie tiež odkladajú preklad do poslednej chvíle, aby pri ňom mohli využiť čo najviac informácií.

Častým porušením YAGNI je snaha programátorov o nadbytočnú všeobecnosť. Napr. ak nie je isté, že program bude musieť spolupracovať s rôznymi databázovými systémami, odporúča sa urobiť ho špecifickým pre konkrétny použitý databázový systém. Udržiavanie jednoduchosti programu zabezpečí, aby zmena databázového systému v budúcnosti nebola náročná.

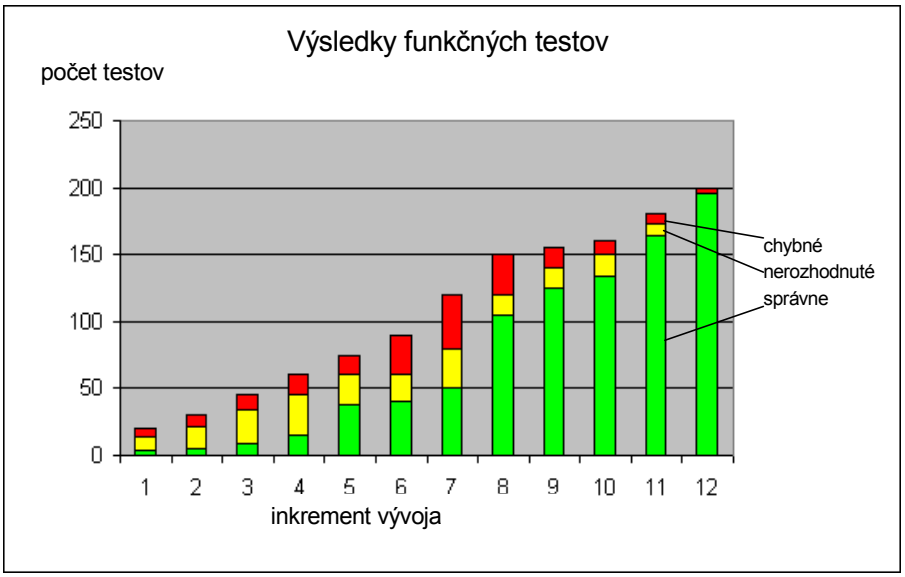
- **Testovanie systému (Functional Tests)**

Testy systému sú programy (alebo dávky), pomocou ktorých sa dá otestovať splnenie externých požiadaviek na funkcionalitu systému, ako aj iných požiadaviek (napr. výkon) [3]. Patria medzi ne programy simulujúce používateľa komunikujúceho s GUI. Automatizácia týchto testov (možno ich tiež označiť za akceptačné testy) umožňuje programovanie zefektívniť a skvalitniť.

Na rozdiel od testov súčiastok, ktoré overujú správnosť súčiastky (obyčajne jednej triedy v OOP), tieto testy overujú celý systém. Testy súčiastok sú vo

vlastníctve programátorov, kým testy systému sú určené zákazníkom. Pomocou nich sa môžu presvedčiť, že ich požiadavky („príbehy“) sú správne implementované [3].

Na sledovanie postupu projektu sa dá použiť miera počtu funkčných testov a zlepšovanie podielu ich úspešnosti. Množina testov počas vývoja narastá a všetky sa neustále aplikujú na vyvíjaný systém. (Viď Obr. 12.)



Obr. 12 Príklad grafu vývoja testov systému [3]

Aby bolo možné testy neustále spúšťať, je nevyhnutné čo najčastejšie (denne alebo aj častejšie) vykonávať integráciu aktuálnej verzie všetkých súčiastok [3]. Takúto prax spomína a odporúča napr. aj [4].

– **Spolupracujúci zákazník (Onsite Customer)**

Pri XP musí zástupca zákazníka (zástupca používateľov) byť počas celého vývoja k dispozícii vývojovému tímu, aby zodpovedal otázky a určoval priority na podrobnej úrovni. Možnosť vyberať si poradie implementovania „príbehov“ a bez obmedzenia si skúšať aktuálnu verziu produktu pomáhajú získať a udržať dôveru zákazníka. Je tiež výborným prostriedkom na prekonanie zotrvačnosti používateľov tým, že vzbudí silný záujem o nový program oveľa viac ako napr. povinné školenia [3].

- **Plánovacia hra (Planning Game)**

XP sa stavia k problému rizikovosti a neexaktnosti plánovania tým, že formuluje plánovanie ako „hru“ [3]:

Figúrkami sú „príbehy“ (používateľské požiadavky). Každý „príbeh“ sa napíše na kartičku. Má priradenú svoju hodnotu (ako si splnenie príslušnej požiadavky cení používateľ) a náklady (odhad času na splnenie zo strany vývojového tímu).

Cieľom je sprevádzkovať príbehy s čo najväčšou celkovou hodnotou „po dobu trvania hry“.

Hráči sú zákazník a vývojový tím.

Možných ťahov je viacero:

1. Napísanie „príbehu“ zákazníkom, priradenie hodnoty a nákladov.
2. Odhad „príbehu“ – odhadne sa čas na implementáciu „príbehu“ v „ideálnych dňoch“ (vyjadrujúcich množstvo práce, ale nie jej rozvrhnutie v čase). „Príbehy“ nad 3 „ideálne týždne“ treba rozdeliť na menšie a naopak „príbehy“ do 1 „ideálneho týždňa“ treba spojiť do jedného väčšieho. Pre účely zostavenia časového rozvrhu sa „ideálne dni“ násobia číslom 3.
3. Určenie záväzku – zákazník a vývojový tím sa dohodnú, ktoré „príbehy“ sa majú implementovať v najbližšej iterácii vývoja systému. Môže byť „riadené príbehmi“ (zákazník si vyberie množinu „príbehov“ a programátori odhadnú čas, kedy budú hotové) alebo „riadené termínom“ (zákazník určí termín ukončenia a programátori mu ponúknu na výber rôzne množiny „príbehov“, ktoré v tomto čase môžu splniť). Vývojový tím si vybranú množinu usporiada podľa pravidla „najprv hodnota a riziko“, t.j. ako prvé v poradí sa budú vyvíjať „príbehy“ s vyššou hodnotou a s vyšším rizikom.
4. Zotavenie z preťaženia – ak sa počas práce na zvolenej iterácii vývoja ukáže, že práce postupujú pomalšie, ako sa predpokladalo, a termín sa nepodarí stihnúť, treba so zákazníkom dohodnúť zmenšenie počtu „príbehov“.
5. Ďalšie ťahy: Napísanie a okamžité zavedenie „príbehu“ do aktuálneho vývoja, rozdelenie „príbehu“ (ak realizácia celého trvala príliš dlho), spresnenie odhadov.

- **Metafora systému (System Metaphor)**

Tento výraz znamená v XP v podstate architektúru systému [3]. Logická štruktúra hlavných tried a objektov systému by mala odzrkadľovať aplikačnú oblasť podľa momentálne platných požiadaviek. Prvky systému by mali byť pomenované názvami z aplikačnej oblasti, takže im rozumie aj používateľ. Uľahčuje sa tým komunikácia s ním.

– **Neustála integrácia, časté zostavovanie (Continuous Integration, Frequent Releases)**

Časté zostavovanie je dôležité pre neustále testovanie systému (viď vyššie), ale aj pretože v XP neexistuje individuálne vlastníctvo kódu, t.j. každý programátor môže zmeniť ktorúkoľvek časť (triedu v OOP) zdrojového kódu celého programu, takže zostavovanie celého systému slúži aj na synchronizáciu zásahov jednotlivých programátorov. Používajú sa nástroje na správu konfigurácií (napr. Microsoft Visual Source Safe).

XP uprednostňuje postup vývoja programu po malých krokoch. Je lepšie pridávať 3 funkcie po jednej a po pridaní každej overiť správnosť súčiastok a systému, ako pridať všetky naraz, pretože s počtom naraz vykonaných zmien sa rýchlo zvyšuje zložitosť ich možnej interakcie.

– **Kolektívne vlastníctvo kódu (Collective Code Ownership)**

Každý programátor môže zasiahnuť do hociktorej časti zdrojového kódu vyvíjaného systému. Súčasne sú však všetci zodpovední za to, aby boli testy súčiastok vždy úspešné.

Predpoklady realizovateľnosti [3]:

- všetci programátori píšu programy podľa spoločných konvencií
- použitie nástrojov na detekciu a riešenie konfliktov
- súprava testov súčiastok na zabezpečenie akosti
- nástroje na preštruktúrovanie programu s automatickou výmenou odkazov na menené časti
- častá integrácia

Je pravdepodobné, že úplné kolektívne vlastníctvo kódu môže dobre fungovať iba v malých tímoch. Tento problém možno riešiť rozdelením väčších tímov na skupiny, v rámci ktorých kolektívne vlastníctvo kódu funguje.

– **Programátorské konvencie (Coding Conventions)**

Jednotný spôsob pomenovávania identifikátorov a formátovania zdrojového kódu jej potrebný, aby mohlo úspešne fungovať kolektívne vlastníctvo kódu. Možno sa pri jeho zavádzaní stretnúť u programátorov s prejavmi odporu [3]. Nielen programátorské konvencie, ale aj iné myšlienky a princípy XP musia byť v tíme všeobecne akceptované. Tímový duch je veľmi potrebný a je lepšie, keď sa programátori, ktorí sa mu nechcú prispôsobiť, na projektoch XP nezúčastňujú.

Záver

Po rozhovoroch s inými softvérovými inžiniermi usudzujem, že zďaleka nielen mňa ako pisateľa tohto článku myšlienka metodiky extrémneho programovania hneď od začiatku očarila. Vo svojej „extrémnosti“ sa zameriava práve na veci, ktoré rezonujú v mysliach tých, čo už majú za sebou nejeden fungujúci program. Obsahuje v sebe „reformátorský“ odkaz, že lepšie je robiť v každej chvíli menej jednoduchých vecí poriadne ako naraz mnoho veľkých a zložitých polovičato.

Takýto prístup môže byť veľmi vhodný najmä pre menšie softvérové projekty, na ktorých sa podieľa, odhadujem, okolo tucta softvérových inžinierov, čiže keď treba minimalizovať réžiu použitého procesu, ale zachovať jeho viditeľnosť. Redukcia počtu rôznych prvkov, z ktorých je model procesu zostavený, prináša elegantnú a ľahko riaditeľnú homogenitu. Na druhej strane do viditeľnosti sa investuje striktné vyžadovanie krátkych vývojových cyklov a dôsledné testovanie každej časti.

Extrémne programovanie je tiež silnou zbraňou v zápase s nejasnými alebo meniacimi sa požiadavkami používateľov. Tým, že explicitne vyžaduje neustálu komunikáciu s používateľom a odsúhlasovanie po malých krokoch, môže významne znížiť riziko softvérového projektu. To má opäť význam najmä pri menších projektoch, keď je neefektívne vopred podrobne špecifikovať požiadavky a túto špecifikáciu zahrnúť do zmluvy medzi zákazníkom a dodávateľom. Pre menšie softvérové firmy môže byť problematické vynútiť si dostatočne podrobné znenie zmluvy. Na druhej strane pre zákazníka je výhodnejšie, ak môže svoje požiadavky v priebehu vývoja meniť.

Tento model procesu vznikol „zdola nahor“ zovšeobecnením praktických skúseností jeho tvorcov. Niektoré jeho princípy bude preto treba podrobiť hlbšiemu teoretickému skúmaniu, aby sa dal presne určiť rozsah ich aplikovateľnosti. Najvýznamnejším rozhodujúcim parametrom je veľkosť projektu.

Vo väčších projektoch zrejme nie je možné dodržiavať v čistej podobe napr. princíp kolektívneho vlastníctva kódu. Je tiež pomerne vágne zadefinované, ako vytvoriť posúdiť „metaforu systému“, najmä pri veľmi veľkých systémoch, u ktorých aj zjednodušené náčrty architektúry môžu byť príliš zložité na jednoznačný výber „najjednoduchšieho“ riešenia. Diskutovať možno o význame programovania v pároch a programátorských konvenciách. V dostupných zdrojoch mi tiež chýbalo podrobnejšie venovanie sa otázke štruktúry dát systémom spracúvaných (napr. databáza) a jej zmeny v súvislosti so zmenami v programe.

V súhrne teda možno odporučiť XP ako zaujímavú alternatívu pre menšie softvérové produkty alebo systémy rozdelené do viac pomerne voľne prepojených častí. Využitie XP vo všeobecnejšom rámci vyžaduje najprv vyjasnenie a zovšeobecnenie niektorých princípov.

Použitá literatúra

[1] Kent Beck: Embracing Change with Extreme Programming. IEEE Computer, Vol. 32, No. 10, October 1999, str. 70-77.

[2] Mária Bieliková: Softvérové inžinierstvo. Princípy a manažment. Slovenská technická univerzita, Bratislava 2000. (ISBN 80-227-1322-8)

[3] Rôzni prispievatelia: Extreme Programming, Unit Tests, Pair Programming, Refactor Mercilessly, You Aren't Gonna Need It, Functional Tests, Onsite Customer, Planning Game, System Metaphor, Frequent Releases, Collective Code Ownership, Coding Conventions. Portland Pattern Repository. April 14, 2000. (URL http://c2.com/cgi/wiki?ExtremeProgramming a odkazy na stránke)

[4] Jim McCarthy: Softwarové projekty. Computer Press, 1999. (ISBN 80-7226-164-0)

Unified Process

Kamil Krnáč

Abstrakt. Unified process umožňuje pozerať sa na vývoj softvéru iteratívne, s väčšou mierou priblíženia sa k používateľovi a s väčším dôrazom na jeho architektúru použitím modelovacieho jazyka UML. Proces je delený na vývojové fázy a ich iterácie, pričom každý dokončený prechod iteračnou cestou znamená absolvovanie všetkých etáp vývoja softvéru (analogických s etapami vodopádového modelu) a vytvorenie funkčného a reálneho evolučného produktu. Fázy procesu sú charakterizované prechodovými podmienkami – „míľnikmi“, ktoré určujú priebeh postupu procesu v časovom a obsahovom pláne.

1. Čo je „Unified Process“?

Unified Process (ďalej RUP) je proces softvérového inžinierstva, ktorý poskytuje komplexný pohľad na prepojenie úloh a zodpovedností v rámci vývojárskej organizácie. Jeho cieľom je zabezpečenie produkcie sotfvéru na vysokej úrovni, ktorý spĺňa požiadavky koncového používateľa v rámci predpokladaného časového a finančného projektového plánu. Namiesto štandardnej produkcie množstva dokumentácie sa sústreďuje na tvorbu a údržbu modelov reprezentujúcich softvér v určitom štádiu vývoja. Je takisto návodom, ako efektívne používať rozšírený jazyk pre modelovanie vývoja softvérových systémov (Unified Modeling Language - UML) a zahŕňa množstvo praktických skúseností z vývojov softvérových systémov vo forme vhodnej pre použitie v širokom spektre typov projektov resp. organizácií.

2. Koncepcia

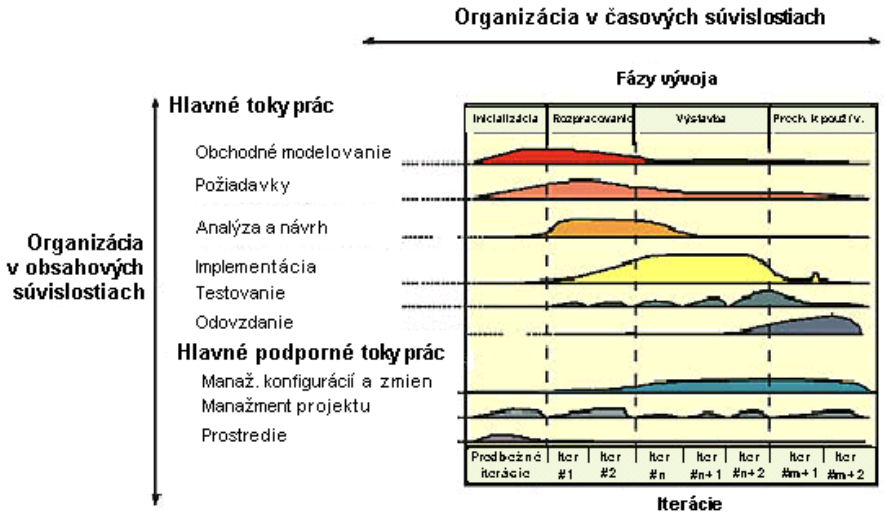
Efektívnu tvorbu softvéru charakterizujú tieto pohľady:

1. *Iteratívny vývoj softvéru*
   - v dnešných softvérových systémoch nie je možné striktne sekvenčná definícia celého vývoja systému. RUP podporuje iteratívne priblíženie sa k uspokojivejšiemu splneniu požiadaviek koncového používateľa pri zníženom riziku neúspechu
2. *Správa požiadaviek*
   - RUP ukazuje, ako je možné organizovať požiadavky a ohraničenia na budúci systém, pričom ich sledovanie a vývoj je prehľadný a efektívne koordinovaný

3. *Architektúra komponentov*
   - RUP popisuje, ako navrhnúť robustný a stabilný systém, ktorý dokáže dostatočne flexibilne reagovať na požiadavky na zmeny a pritom je ale intuitívne pochopiteľný a poskytuje nástroje pre softvérové znovupoužitie
4. *Vizuálne modelovanie softvéru*
   - RUP pomocou grafických stavebnicových blokov vizualizuje štruktúru a správanie sa komponentov systému resp. celej architektúry a zlepšuje tak prehľadnosť a komunikáciu prvkov vývoja systému
5. *Verifikácia kvality softvéru*
   - spoľahlivosť je jedným z faktorov, ktoré bránia vyššej akceptácii softvérových systémov dnešnej doby. RUP pomáha pri plánovaní, návrhu, implementácii, vykonávaní a zhodnotení testov kvality, pričom tieto testy sú včlenené do procesu ako nedeliteľná súčasť a prebiehajú v každej etape vývoja softvéru
6. *Riadenie zmien*
   - zabezpečuje mechanizmy na zaručenie riadenia akceptácie zmien a monitorovanie ich realizácie, čo je nutnou súčasťou iteratívneho vývoja, ktorý RUP implicitne pokrýva.

3. Priebeh procesu

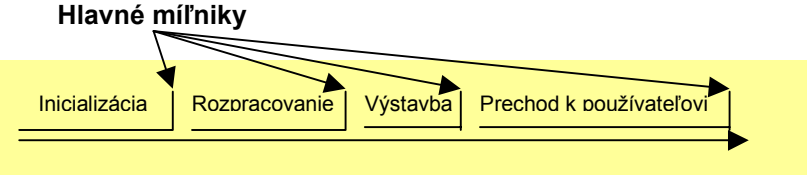
Proces tvoria 4 fázy a 9 etáp (Obr. 13) [2]. Každá fáza by mala pozostávať z iterácií (viacnásobný prechod tou istou fázou), čím sa redukujú riziká komunikácie medzi vývojármi a zákazníkmi, pretože na konci každej iterácie vzniká buď interný alebo externý reálny funkčný systém (nie finálny). Tento fakt spôsobuje taktiež vyššiu motiváciu vývojárov kvôli existujúcim blízkym relatívne ľahko dostupným cieľom, čo sa pozitívne odráža na pružnejších reakciách na požiadavky, a teda aj na dodržaní termínov dodávky systému zákazníkovi.



Obr. 13 Fázy, iterácie a etapy priebehu procesu RUP [2].

Horizontálna os grafu na Obr. 13 určuje fázy, cykly, iterácie a míľniky (časové súvislosti) a vertikálna os činnosti, zdroje a výstupy (obsahové súvislosti).

Jednou z vlastností RUP cielenou na zníženie rizika spočíva v existencii rozhodovacích bodov na konci každej iterácie – tzv. "míľnik". Splnenie resp. nesplnenie požiadaviek a cieľov definovaných pre danú fázu procesu znamená buď pokračovanie procesu ďalšou fázou, vykonanie ďalšej iterácie v aktuálnej fáze, alebo úplné pozastavenie projektu (viď Obr. 14) [2].



Obr. 14 Fázy a hlavné míľniky v procese RUP [2].

Iterácia nazývaná podľa niektorých zdrojov aj tzv. mini-projekt [1] je akýmsi priebehom všetkých etáp vývoja (analógia s etapami vodopádového modelu), pričom odhalená chyba alebo nesprávny prístup v rámci jednej iterácie (čo má za následok neplánované zopakovanie jednej iterácie a nie celého procesu) má výrazne menší vplyv na pokračovanie vývoja projektu v porovnaní s klasickým jednoprechodovým procesom. Tento prístup má výhodu vo včasnom odhalení zlomových bodov, ktoré

spôsobujú až 85% mieru chybovosti koncepcií vyvíjaných priemyselných softvérových systémov [3].

Fáza inicializácie (angl. *inception phase*)

V tejto fáze je zadefinovaný rozsah projektu, sú identifikované všetky externé entity a definované interakcie medzi nimi a systémom. Zahŕňa kritériá úspešnosti, riziká, odhad potrebných zdrojov a rozvrh so zadefinovanými časovými míľnikmi.

Výstupom je vízia, základný model prípadov použitia (angl. *use-case model*) pokrývajúci definované procesy systému smerom k používateľovi, plány (rozvrh, rozsah...), zhodnotenie rizík, jeden alebo viac prototypov.

Míľnik: (voči zákazníkovi) Dohoda o rozsahu, pochopenie požiadaviek, odsúhlasenie časových a rozpočtových plánov, architektonického pohľadu (prototyp), rozšírení.

Fáza rozpracovania (angl. *elaboration phase*)

Prebieha analýza problémovej domény, tvorba projektového plánu, eliminácia najvýznamnejších rizík. Táto fáza zaisťuje, že architektúra, požiadavky a plány sú dostatočne stabilné, takže je možné dodefinovať časové a rozpočtové plány celého projektu. Vytvára sa funkčný prototyp architektúry (kostra s minimom programového vybavenia na podporu funkčnosti).

Výstupom je takmer kompletný model prípadov použitia (všetky prípady a používateľské úlohy sú identifikované), popis SW architektúry, spustiteľný architektonický prototyp, zoznam rizík, celkové projektové plány, predbežné používateľské príručky.

Míľnik: Stabilná vízia a architektúra, známe riešenia významných rizík, dostatočný a detailný plán pre konštrukčnú fázu, komplexné odsúhlasenie plánu zákazníkom.

Fáza výstavby (angl. *construction phase*)

Počas tejto fázy prebieha reálna produkcia a integrácia systému s možnosťou paralelných činností pri väčších systémoch.

Výstup je softvérový produkt integrovaný na danú platformu, používateľské príručky a popis súčasnej verzie (produkt je v tejto fáze nazývaný “beta” verziou).

Míľnik: Produkt je stabilný a pripravený na prechod do reálneho prostredia.

Fáza prechodu k používateľovi (angl. *transition phase*)

Cieľom tejto fázy je aplikácia produktu v reálnom prostredí. Dostupnosť produktu koncovému používateľovi totiž vyústi do spätnoväzobných podnetov vo

forme požiadaviek na zmeny, ktoré sa zakomponujú do ďalších verzií. Zahŕňa teda "beta" testovanie, úpravy častí systému vzhľadom na zmeny, školenia používateľov a marketingové, distribučné a obchodné činnosti.

Míľnik: Používateľ je spokojný a rozšírenia sú voči plánovaným akceptovateľné.

4. Statická štruktúra procesu

RUP definuje vo svojej štruktúre tieto časti [2]:

- Roly (angl. *workers*) – definuje správanie sa resp. zodpovednosť používateľa alebo skupiny vzhľadom k systému (úloha)
- Činnosti (angl. *activities*) – elementárna jednotka práce – činnosť – viazaná k špecif. úlohe
- Entity (angl. *artifacts*) – entity informácie tvorené, modifikované a používané v procese (modely, plány, zdrojové kódy, dokumentácie ...)
- Toky prác (angl. w*orkflows*) – schémy väzieb a postupností úloh, činností a ich výsledkov (etapy)

V priebehu procesu RUP (Obr. 13) sú naznačené tieto etapy:

- Obchodné modelovanie (angl. *Business Modeling*) – definuje jednotný komunikačný jazyk medzi komunitou softvérových inžinierov a inžiniermi obchodnej komunity
- Požiadavky (angl. *Requirements*) – súhrn požiadaviek na prácu systému smerom ku používateľovi („case" model). Každý „prípad" je detailne definovaný.
- Analýza a návrh (angl. *Analysis & Design*) – popis, ako má byť systém realizovaný v etape implementácie – cieľom je navrhnúť dostatočne robustný a požiadavkám vyhovujúci systém
- Implementácia (angl. *Implementation*) – realizácia systému prostredníctvom integrácie implementovaných komponentov
- Testovanie (angl. *Test*) – verifikácia správnej integrácie, interakcie objektov, splnenia požiadaviek, korektného spracovania prípadných defektov
- Odovzdanie (angl. *Deployment*) – úspešná produkcia finálneho produktu a doručenie koncovému používateľovi (zákazníkovi) vrátane inštalácií, konzultácií a migrácie údajov medzi rôznymi produktmi alebo produktmi rôznych verzií
- Manažment konfigurácií a zmien (angl. *Configuration & Change Management*) – rieši organizáciu veľkého množstva výstupov z procesu s ohľadom na organizáciu verzií a na distribuovaný (a zároveň možný súčasný) prístup k nim
- Prostredie (angl. *Environment*) – poskytnutie procesov a nástrojov pre vývojárske organizácie, ktoré sú potrebné pre podporu ich vývojárskych tímov vzhľadom na projektové riadenie práce

5. Záver

Použitie RUP v praxi má svoje silné i slabé stránky. Výhodná je jeho orientácia na iteratívny vývoj silne ovplyvnený požiadavkami pričom jeho základom je architektúra softvéru. Poskytuje mechanizmy na tvorbu prototypov na konci každej iterácie, rozhodovanie o pokročení resp. nepokročení v priebehu procesu vývoja, čo umožňuje zviditeľnenie stavu projektu pre potreby riadenia [4]. Výhody využívania iterácií spočívajú vo včasnom odhalení a eliminácií rizík, lepšom zvládnutí požadovaných zmien, vyššom stupni znovupoužitia, v možnosti profesionálneho rastu projektového tímu počas celého priebehu vývoja a v iteratívnym prístupom možno dosiahnuť vyššiu celkovú kvalitu produktu [1].

Medzi slabé stránky RUP patrí napríklad fakt, že RUP je iba procesom vývoja softvéru a nezahŕňa celý životný cyklus vrátane koncepcie údržby a podpory, nepodporuje vývoj multi-projektov zahŕňajúcich potrebu znovupoužitia v rozsiahlejších organizačných štruktúrach, neposkytuje dostatok podporných nástrojov pre jeho realizáciu v praxi a zaostáva v pokrytí oblastí manažmentu metrík, znovupoužitia, ľudských zdrojov a testovania [3,4].

6. Použitá litaratúra

[1] Ivar Jacobson, Grady Booch, and Jim Rumbaugh: The unified process. IEEE Software, May/June 1999, str. 96 – 102.

[2] Rational Unified Process: Best Practices for Software Development Teams. A Rational Software Corporation White Paper, 13. 4. 2000. (URL http://www.rational.com/dynamic/whitepapers/media/rup_bestpractices.pdf)

[3] Scott W. Ambler, Enhancing the Unified Process: Software Process for the Post-2000 (P2K) World, A Ronin International White Paper, 15. 3. 2000. (URL http://www.ronin-intl.com/unifiedProcess.PDF)

[4] W. Ambler, Completing the Unified Process With Process Patterns, 13. 4. 2000. (URL http://www.ambysoft.com/unifiedProcess.html)

Táto kniha bola pripravená na tlač v elektronickej podobe. Text bol napísaný v textovom editore Microsoft Word 97 pre Windows. Pri písaní sa použili rôzne štýly ostavcov (Obsah, Názov, Adresa, Abstrakt, Nadpis 2, Nadpis 3, Referencia, Normal). Pri návrhu grafickej podoby publikácie sa použilo písmo Times New Roman.

Autori esejí
SOWA (ako je uvedené pri každej eseji)

Odborní redaktori
Ján Glončák
Róbert Kovaľ
Kamil Krnáč
Michal Kucbel
Branislav Lehotský
Peter Liczki
Marián Varga

Príbeh k esejám, návrh obálky, výroba CD
Róbert Kovaľ

Vedenie kolektívu, grafické úpravy
Viktor Zeman

Korektúry
Viktor Zeman, Marián Varga

Integrácia
Richard Koncz, Viktor Zeman

Tlač, väzba
Róbert Kovaľ

Prezentácia
Richard Mátéffy

Názov SOWA - Etika softvérového inžiniera a softvérové procesy
Autori Bc. Ján Glončák, Bc. Róbert Kovaľ, Bc. Kamil Krnáč,
Bc. Richard Koncz, Bc. Michal Kucbel,
Bc. Branislav Lehotský, Bc. Peter Liczki,
Bc.Richard Mátefy, Bc. Marián Varga, Bc. Viktor Zeman

Vydalo SOWA, Ilkovičova 3, Bratislava
Tlač, väzba Róbert Kovaľ
Vydanie prvé
Náklad 1 výtlačok
Rozsah 123 strán