Marcel Bočkay – Peter Burger – Matej Černák
Roman Ďumbier – Robert Dvoran – Roman Filkorn
Jaroslav Hanúsek – Peter Jehlár – Tomáš Milička
Filip Pucher – Branislav Šimo

SOFTVÉROVÉ INŽINIERSTVO

ETICKÝ KÓDEX & PROCESY VÝVOJA SOFTVÉRU

SOFTVÉROVÉ INŽINIERSTVO

ETICKÝ KÓDEX A PROCESY VÝVOJA SOFTVÉRU

Marcel Bočkay – Peter Burger – Matej Černák
Roman Ďumbier – Robert Dvoran – Roman Filkorn
Jaroslav Hanúsek – Peter Jehlár – Tomáš Milička
Filip Pucher – Branislav Šimo

SOFTVÉROVÉ INŽINIERSTVO

ETICKÝ KÓDEX & PROCESY VÝVOJA SOFTVÉRU

Slovenská technická univerzita
2000

Bc. Marcel Bočkay
Bc. Matej Černák
Bc. Robert Dvoran
Bc. Jaroslav Hanúsek
Bc. Tomáš Milička
Bc. Branislav Šimo
Bc. Peter Burger
Bc. Roman Ďumbier
Bc. Roman Filkorn
Bc. Peter Jehlár
Bc. Filip Pucher

Softvérové inžinierstvo

Etický kódex a procesy vývoja softvéru

© GWAIN 2000

V knihe použité názvy programových produktov, firiem apod. môžu byť ochrannými známkami alebo registrovanými ochrannými známkami príslušných vlastníkov.

Kniha neprešla jazykovou úpravou.

Venujeme doc. Ing. Márii Bielikovej, PhD., ktorá iniciovala vytvorenie tejto knihy, ale aj všetkým pracovníkom Katedry informatiky a výpočtovej techniky FEI STU Bratislava, ktorí priamo alebo nepriamo ovplyvnili našu cestu životom.

autori

Predslov

Publikácia, ktorú práve držíte v rukách, tvorí výsledok našej práce na predmete Manažment v softvérovom inžinierstve. Nájdete v nej mnoho cenných myšlienok a postrehov z oblasti dodržiavania etických princípov softvérových inžinierov, ako aj pohľad na niektoré nové modely vývoja softvérových produktov. Naša publikácia je určená študentom zaoberajúcim sa manažmentom softvérového inžinierstva, ktorí v nej nájdu mnoho užitočných faktov, využiteľných počas svojho štúdia. Všetky uvedené články vychádzajú z množstva odbornej literatúry, ktorú sme mali možnosť počas semestra prečítať, preto táto publikácia bude určite prínosom aj pre tých, ktorí hľadajú nové poznatky z oblasti aplikovanej etiky a modelov softvérových systémov.

Naša publikácia je rozdelená do dvoch nezávislých častí. V prvej polovici sme sa zamerali na etický kódex softvérového inžiniera. Vo forme esejí prinášame pohľad na niektoré morálne problémy, s ktorými sa počítačoví odborníci počas svojej praxe stretnú. Druhá časť knižky obsahuje technické články, v ktorých sme sa snažili načrtnúť niektoré nové trendy v navrhovaní softvéru. Ako vhodný doplnok odporúčame elektronické médium k tejto publikácii, kde okrem uvedených prác nájdete aj prezentačné materiály k článkom a videozáznam z našej prezentácie.

Na tomto mieste by sme chceli poďakovať Doc.Ing. Márii Bielikovej, PhD., ktorá vedie predmet Manažment v softvérovom inžinierstve na Fakulte elektrotechniky a informatiky STU Bratislava – za jej ochotu, nápady a cenné rady pri tvorbe našich prác. Zároveň by sme sa chceli poďakovať všetkým našim kolegom na tomto predmete, ktorí nám konštruktívnou kritikou pomáhali počas uplynulých dní.

GWAIN
(Group Without An Interesting Name)

Obsah

DIEL I. Eseje na tému: Etický kódex softvérového inžiniera

ÚVOD 3

KÓDEX ETIKY A PROFESIONÁLNEJ PRAXE SOFTVÉROVÉHO INŽINIERSTVA 5

ETIKA AKO ZABEZPEČENIE KVALITY? 13

JA, ON A ON 19

POTULKY ŽIVOTOM ALEBO S ČÍM SA V ŽIVOTE STRETÁVAME 25

MAMI, JA SOM NECHCEL...! 31

HLBŠÍ VÝZNAM 37

SÚDNOSŤ V SOFTVÉROVOM INŽINIERSTVE, ALEBO „LINUX VERSUS WINDOWS“ 45

ETIKA AKO PRINCÍP POZITÍVNYCH HODNÔT 51

BLIŽŠIA KOŠEĽA AKO KABÁT 55

KOLEGOVIA – PROBLÉMY A RIEŠENIA 61

DRAČIE SRDCE 67

Z VNÚTRA PRAMENÍ SILA 73

ZÁVER 77

DIEL II. Odborné články na tému: Nové trendy v procesoch vývoja softvéru

ÚVOD 81

SOFTVÉROVÉ PROCESY 83

KLASICKÉ MODELY 87

EVOLUČNÝ MODEL 93

KOMPONENTVÉR 97

NÁVRHOVÉ VZORY A ICH PRÍNOS 105

MODULÁRNY SOFTVÉROVÝ PROCES METÓDOU MINI-OHODNOTENIA 111

ÚSPECH SOFTVÉROVÝCH PROJEKTOV – KRITICKÉ FAKTORY 119

EXTRÉMNE PROGRAMOVANIE 125

SYNCHRONIZUJ A STABILIZUJ 133

OPEN SOURCE 141

CERTIFIKAČNÉ TESTOVANIE 151

ZÁVER 89

PRÍLOHA ELEKTRONICKÝ NOSIČ (CD-ROM)

DIEL I.

Eseje na tému:

Etický kódex softvérového inžiniera

Úvod

Tak, ako v súčasnej dobe sledujeme obrovský rozmach informačných technológií, zároveň sa stávame aj svedkami prieniku etiky čoraz viac a viac do každodenného života – čoraz viac a viac do každodennej práce. Mnohé spoločenstvá, či už sú to produktívne firmy, záujmové alebo odborné skupiny, zistili, že etické princípy sú neoddeliteľnou súčasťou ich práce. Zaviedli a snažia sa dodržiavať etické kódexy, lebo vidia, že to má pozitívny prínos k efektivite práce v kolektíve.

Prostredníctvom etických kódexov inštitúcia, firma či iná skupina ľudí dáva svojim členom i okoliu na vedomie, že etika sa stala nevyhnutnou súčasťou jej činnosti. Etický kódex prezentuje skupinové, kolektívne normy a princípy, ktoré by mali usmerňovať činnosť každého člena skupiny. Každé indivíduum sa potom nachádza pod vplyvom nielen vlastného morálneho presvedčenia, ale aj pod vplyvom spoločenských morálnych pravidiel. Často sa stretávame s problémom, keď sa etický princíp dostáva do konfliktu s ekonomickými alebo odbornými princípmi. Etický kódex by nemal sťažovať výkon odbornej funkcie, ale mal by napomôcť rozhodovaniu členov profesie alebo pracovníka firmy s ohľadom na určité morálne hodnoty. Etický kódex pomáha riešiť konfliktné situácie a uľahčuje odborníkovi rozhodnúť sa v situáciách, keď sa nevie zorientovať medzi tým, čo je správne a čo nesprávne, resp. vtedy keď je vedený inými požiadavkami (finančnými, prestížnymi...) vykonať nemorálny skutok.

Etický kódex softvérových inžinierov obsahuje sedem skupín princípov, ktoré si kladú za úlohu inšpirovať a viesť profesionálov k lepšiemu vzťahu k etickým problémom. Jeho najdôležitejšie body sú: vzťah k verejnosti, k zamestnancom a k zamestnávateľovi, vzťahy medzi kolegami navzájom, dodržiavanie kvality, zásada odbornosti a ďalšie.

V nasledujúcich esejach, uvádzame niekoľko pohľadov na dodržiavanie etického kódexu softvérových inžinierov v praxi a priblížime Vám niekoľko problémov a konfliktov, ktoré sa dodržiavaním týchto pravidiel môžu vyskytnúť. Pre úplnosť uvádzame aj kompletné znenie etického kódexu softvérových inžinierov podľa IEEE.

Kódex etiky a profesionálnej praxe softvérového inžinierstva

IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices[1]

Preambula

Počítače majú ústrednú a stále sa zväčšujúcu úlohu v obchode, priemysle, vláde, zdravotníctve, zábave a spoločnosti. Softvéroví inžinieri sú tými, ktorí prispievajú priamou účasťou alebo vyučovaním k analýze, špecifikácii, návrhu, vývoju, certifikácii, údržbe a testovaniu softvérových systémov. V dôsledku svojich funkcií pri vývoji softvérových systémov majú softvéroví inžinieri významné možnosti konať dobro alebo spôsobiť škodu, umožniť iným konať dobro alebo spôsobiť škodu, alebo ovplyvniť iných, aby konali dobro alebo spôsobili škodu. Aby sa v maximálnej možnej miere zaistilo, že ich úsilie bude použité za dobrým účelom, musia sa softvéroví inžinieri zaviazať, že budú konať tak, aby pozdvihli softvérové inžinierstvo na prospešnú a váženú profesiu. V súlade s týmto záväzkom by softvéroví inžinieri mali dodržiavať nasledovný Kódex etiky a profesionálnej praxe softvérového inžinierstva.

Kódex obsahuje osem Princípov súvisiacich so správaním sa a rozhodnutiami vykonávanými profesionálnymi softvérovými inžiniermi, zahŕňajúc vykonávateľov praxe, pedagógov, manažérov, vedúcich pracovníkov a tvorcov koncepcií, ako aj praktikantov a študentov profesie. Princípy identifikujú eticky zodpovedné vzťahy, ktorých sa zúčastňujú jednotlivci, skupiny a organizácie, a najhlavnejšie záväzky v rámci týchto vzťahov. Odstavce každého Princípu sú ilustráciou niektorých záväzkov zahrnutých v týchto vzťahoch. Tieto záväzky majú základ v humanite softvérového inžiniera, v mimoriadnom ohľade, ktorý je povinný vo vzťahu k ľuďom ovplyvneným prácou softvérových inžinierov, a špecifickými prvkami praxe softvérového inžinierstva. Kódex ich predpisuje ako záväzky každého, kto sa prehlasuje byť alebo sa snaží byť softvérovým inžinierom.

Nie je zámerom, aby boli jednotlivé časti Kódexu použité izolovane na ospravedlnenie nedostatkov alebo výšky odmeny. Zoznam Princípov a Odstavcov nie je vyčerpávajúci. Odstavce by sa nemali chápať ako oddelenie akceptovateľného od neakceptovateľného v profesionálnom správaní sa vo všetkých praktických situáciách. Kódex nie je

---

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

jednoduchým etickým algoritmom generujúcim etické rozhodnutia. V niektorých situáciách môžu byť štandardy v rozpore so sebou navzájom alebo so štandardami z iných zdrojov. Tieto situácie vyžadujú od softvérového inžiniera použitie etického úsudku, aby za daných okolností jednal spôsobom najviac zhodným s duchom Kódexu etiky a profesionálnej praxe.

Etické rozpory je možné najlepšie riešiť pozorným uvážením základných princípov namiesto slepého spoliehania sa na detailné predpisy. Tieto Princípy by mali ovplyvniť softvérových inžinierov, aby celkovo zvážili, kto ovplyvnený ich prácou, preverili, či oni a ich kolegovia jednajú s inými ľudskými bytosťami s príslušným rešpektom, zvážili, ako by na ich rozhodnutia pozerala dostatočne dobre informovaná verejnosť, analyzovali, ako budú tí najmenej splnomocnení zasiahnutí ich rozhodnutiami, a zvážili, či by ich skutky boli posudzované ako hodné dokonalého profesionála pracujúceho ako softvérový inžinier. Pri všetkých týchto rozhodnutiach je primárnym záujem zdravia, bezpečnosti a prospechu verejnosti, t. j. "Verejný záujem" je hlavným pre tento Kódex.

Dynamický a náročný kontext softvérového inžinierstva vyžaduje kódex, ktorý je prispôsobivý a platný vzhľadom na vyskytnuvšie sa nové situácie. Avšak aj napriek tejto všeobecnosti Kódex poskytuje podporu softvérovým inžinierom a manažérom softvérových inžinierov, ktorí potrebujú v konkrétnom prípade konať pozitívne, dokumentovaním etického stanoviska profesie. Kódex poskytuje etický základ, na ktorý sa môžu jednotlivci v tíme ako aj celé tímy odvolávať. Kódex pomáha definovať skutky, ktoré je eticky nesprávne vyžadovať od softvérového inžiniera alebo tímu softvérových inžinierov.

Kódex nie je len na posudzovanie povahy pochybných činov, má taktiež dôležitú vzdelávaciu funkciu. Tým, že tento Kódex vyjadruje konsenzus profesie v otázkach etiky, je prostriedkom k vzdelaniu verejnosti ako aj ašpirujúcich profesionálov v oblasti etických záväzkov celého softvérového inžinierstva.

Princípy

Princíp 1: Verejnosť

Softvéroví inžinieri by mali konať v zhode so všeobecnými verejnými záujmami. Obzvlášť by softvéroví inžinieri mali:

1.01 Prijať plnú zodpovednosť za vlastnú prácu.

1.02 Usmerňovať záujmy a požiadavky softvérového inžiniera, zamestnávateľa, klienta a používateľov, aby boli v zhode s verejným prospechom.

1.03 Schváliť softvérový produkt iba v prípade, ak sú odôvodnene presvedčený o tom, že produkt je bezpečný, spĺňa špecifikačné požiadavky, neznižuje kvalitu

života, neohrozuje súkromie a životné prostredie a úspešne vyhovel príslušným testom. Hlavným výsledkom práce by malo byť všeobecné blaho.

1.04 Prezentovať vhodnej osobe alebo autorite každé existujúce alebo potenciálne ohrozenie používateľa, verejnosti alebo životného prostredia, o ktorom je odôvodnene presvedčený, že je spôsobené alebo spojené so softvérom alebo príslušnými dokumentmi.

1.05 Podporovať úsilie, ktoré sa zaoberá záležitosťami s vážnym spoločenským dopadom, spôsobenými softvérom, jeho inštaláciou, údržbou, podporou a dokumentáciou.

1.06 Byť úprimný a vyhýbať sa klamu pri všetkých svojich vyhláseniach, obzvlášť na verejnosti, ktoré sa týkajú softvéru alebo príslušných dokumentov, metód a nástrojov.

1.07 Zvažovať otázky spojené s fyzickými obmedzeniami, prideľovaním prostriedkov, ekonomickými nevýhodami a inými faktormi, ktoré by mohli obmedziť využívanie výhod poskytovaných softvérom.

1.08 Byť povzbudení dobrovoľne poskytovať svoje profesionálne znalosti na dobré ciele a prispievať k zvyšovaniu úrovne znalostí verejnosti v oblasti softvérového inžinierstva.

Princíp 2: Zákazník a zamestnávateľ

Softvéroví inžinieri by mali jednať v najlepších záujmoch svojho zákazníka a zamestnávateľa, v zhode s verejným záujmom. Obzvlášť by softvéroví inžinieri mali:

2.01 Poskytovať služby v oblastiach svojej pôsobnosti, čestne a otvorene informovať o obmedzeniach súvisiacich s ich skúsenosťami a vzdelaním.

2.02 Vedome nepoužívať softvér, ktorý je získaný alebo prechovávaný nelegálne alebo neeticky.

2.03 Používať majetok zákazníka alebo zamestnávateľa len spôsobmi riadne oprávnenými a s vedomím a so súhlasom zákazníka a zamestnávateľa.

2.04 Zabezpečiť, že ktorýkoľvek dokument, na ktorý sa zákazník alebo zamestnávateľ spolieha, bol schválený osobou na to poverenou.

2.05 Udržiavať v tajnosti dôverné informácie získané počas svojej odbornej pôsobnosti, kde dôvernosť je v zhode s verejným záujmom a v zhode so zákonom.

2.06 Identifikovať, dokumentovať, zhromažďovať údaje a oznámiť ich zákazníkom alebo zamestnávateľom okamžite, keď by podľa ich názoru mohol byť projekt neúspešný, preukázateľne drahý, mohol by porušovať zákon duševného vlastníctva, alebo byť ináč problematický.

2.07 Identifikovať, dokumentovať a oznámiť zamestnávateľovi alebo zákazníkovi dôležité problémy spoločenského rozsahu, ktoré si uvedomujú, vzhľadom k softvéru alebo v súvisiacim dokumentom.

2.08 Neakceptovať žiadnu vedľajšiu prácu, ktorá by mala škodlivý vplyv na prácu, ktorú uskutočňujú pre svojho hlavného zamestnávateľa.

2.09 Nepodporovať žiadne záujmy zamerané proti svojmu zamestnávateľovi alebo zákazníkovi, iba ak je to v zhode s vyšším etickým záujmom. V takom prípade by o ňom mali informovať zamestnávateľa alebo inú vhodnú autoritu.

Princíp 3: Výrobok

Softvéroví inžinieri by mali zaistiť, aby ich produkty a súvisiace modifikácie spĺňali najvyšší možný profesionálny štandard. Obzvlášť by softvéroví inžinieri mali:

3.01 Usilovať o vysokú kvalitu, akceptovateľné náklady a zmysluplný plán, zabezpečiac, aby významné kompromisy boli jasné a akceptované zamestnávateľom aj klientom a prístupné stanovisku používateľa aj verejnosti.

3.02 Zabezpečiť stanovenie náležitých a dosiahnuteľných cieľov a zámerov v každom projekte, na ktorom pracujú alebo sa inak spolupodieľajú.

3.03 Identifikovať, definovať a venovať sa etickým, ekonomickým, kultúrnym, právnym a enviromentálnym dôsledkom zviazaným s rozpracovanými projektmi.

3.04 Zabezpečiť svoju kvalifikáciu na každom projekte, na ktorom pracujú alebo sa spolupodieľajú, primeranou kombináciou vzdelania, školení a skúseností.

3.05 Zabezpečiť použitie primeranej metódy pre každý projekt, na ktorom pracujú alebo zamýšľajú pracovať.

3.06 Dodržiavať pri aktuálnej práci primerané profesionálne štandardy vždy, keď je to možné, odchyľujúc sa od nich iba v eticky a technicky oprávnených prípadoch.

3.07 Usilovať o úplné porozumenie špecifikácie softvéru, na ktorom pracujú.

3.08 Zabezpečiť, aby špecifikácia softvéru, na ktorom pracujú, bola dobre zdokumentovaná, spĺňala užívateľove požiadavky a bola náležite schválená.

3.09 Zabezpečiť realistické kvantitatívne odhady nákladov, plánu, osôb, kvality a výstupov na každom projekte, na ktorom pracujú alebo zamýšľajú pracovať a poskytnúť pravdepodobné hodnotenie týchto odhadov.

3.10 Zabezpečiť adekvátne testovanie a kontrolu softvéru a prislúchajúcich dokumentov, na ktorých pracujú.

3.11 Zabezpečiť adekvátnu dokumentáciu, obsahujúcu objavené dôležité problémy a použité riešenia pre každý projekt, na ktorom pracujú.

3.12 Prácou vytvárať softvér a prislúchajúcu dokumentáciu, ktorá rešpektuje súkromie tých, ktorí budú ovplyvnení daným softvérom.

3.13 Byť opatrní a používať iba správne údaje získané zákonnou a etickou cestou a použiť ich iba autorizovanými spôsobmi.

3.14 Udržovať integritu dát, sledujúc najmä zastaralé a poškodené prípady.

3.15 Pristupovať ku všetkým formám softvérovej údržby s rovnakou profesionalitou ako k novému vývoju.

Princíp 4: Posudzovanie

Softvérový inžinieri by mali zachovávať integritu a nezávislosť vo svojom profesionálnom posudzovaní. Obzvlášť by softvéroví inžinieri mali:

4.01 Zmierniť všetky technické posudky potrebou podpory a udržania ľudských hodnôt.

4.02 Schvaľovať iba dokumenty pripravené pod ich dohľadom alebo v rámci oblasti ich kompetencie, s ktorými súhlasia.

4.03 Dodržovať profesionálnu objektivitu so zreteľom na akékoľvek softvérové alebo súvisiace dokumenty pri požiadavke o ich zhodnotenie.

4.04 Nezapájať sa do podvodných finančných praktík, ako úplatkárstvo, dvojité účtovanie alebo iných nevhodných finančných praktík.

4.05 Informovať všetky zúčastnené strany, ktorých konfliktom záujmov nemožno primerane zabrániť alebo vyhnúť sa im.

4.06 Odmietnuť účasť ako členovia alebo poradcovia v súkromných, vládnych, alebo v profesionálnych telesách zaoberajúcich so softvérom alebo s jemu príbuznými aktivitami v ktorých oni sami, ich zamestnávatelia alebo klienti majú nezverejnené potencionálne konflikty záujmov.

Princíp 5: Manažment

Manažéri a lídri softvérového inžinierstva by mali majú podporovať a propagovať etický prístup k manažmentu softvérového vývoja a údržby. Manažéri a vedúci softvéroví inžinieri by obzvlášť mali:

5.01 Zabezpečiť dobrý manažment pre všetky projekty, na ktorých pracujú, vrátane efektívnych procedúr na podporu kvality a znižovania rizika.

5.02 Zabezpečiť, aby softvéroví inžinieri boli informovaní o štandardoch skôr, ako ich majú dodržiavať.

5.03 Zabezpečiť, aby softvéroví inžinieri poznali koncepciu zamestnávateľa a procedúry na ochranu hesiel, súborov a informácií, ktoré sú dôverné pre zamestnávateľa alebo ostatné osoby.

5.04 Zadávať prácu až po zohľadnení vhodných príspevkov vzdelania a skúseností, kombinujúc túžbu rozširovať takéto vzdelanie a skúsenosti.

5.05 Zabezpečiť reálne kvantitatívne odhady nákladov, rozvrhu, personálu, kvality a výsledkov všetkých projektov, na ktorých pracujú alebo navrhujú pracovať, a poskytovať vyhodnotenie neistoty týchto ukazovateľov.

5.06 Pritiahnuť potenciálnych softvérových inžinierov iba úplným a presným opisom podmienok zamestnania.

5.07 Ponúknuť spravodlivé a presné odmeňovanie.

5.08 Nebrániť nespravodlivo niekomu v tom, aby zaujal pozíciu, na ktorú je vhodne kvalifikovaný.

5.09 Zabezpečiť spravodlivé dohody o vlastníctve týkajúce sa každého softvérového, procesného, výskumného, písomného alebo iného duševného vlastníctva, ku ktorému softvérový inžinier prispel.

5.10 Umožniť riadny proces pri konaní o porušení koncepcie zamestnávateľa alebo tohto Kódexu.

5.11 Nežiadať od softvérového inžiniera, aby vykonal to, čo nie je v súlade s týmto Kódexom.

5.12 Nikoho netrestať za vyjadrenie etických pripomienok k určitému projektu.

Princíp 6: Profesia

Softvéroví inžinieri by mali zlepšovať bezúhonnosť a povesť profesie v súlade s verejným záujmom. Obzvlášť by softvéroví inžinieri mali:

6.01 Pomáhať vytvárať pracovné prostredie napomáhajúce etickému správaniu.

6.02 Zvyšovať povedomie verejnosti o profesii softvérového inžiniera.

6.03 Rozširovať znalosť softvérového inžinierstva vhodnou účasťou v profesijných organizáciách, stretnutiach a publikáciách.

6.04 Podporovať, ako členovia profesie, ostatných softvérových inžinierov snažiacich sa o dodržiavanie tohoto kódexu.

6.05 Nepovyšovať vlastné záujmy na úkor profesie, klientov alebo zamestnávateľa.

6.06 Dodržiavať všetky pravidlá týkajúce sa práce softvérových inžinierov, okrem prípadov, keď takýto postoj nie je v zhode s verejnými záujmami.

6.07 Byť presný pri stanovovaní vlastností softvéru, na ktorom pracujú a vyhnúť sa tým stanoveniu nielen nepravdivých tvrdení, ale aj tvrdení špekulatívnych, bezduchých, klamlivých, zavádzajúcich alebo pochybných.

6.08 Preberať zodpovednosť za hľadanie a opravu chýb, za vytváranie správ o chybách v softvéri a v príslušných dokumentoch, na ktorých pracujú.

6.09 Zaistiť, aby klienti, zamestnávatelia a vedúci pracovníci vedeli o záväzku softvérového inžiniera dodržiavať tento Kódex a z toho vyplývajúcich ostatných záväzkov.

6.10 Vyhýbať sa spájaniu s ľuďmi, spoločnosťami a organizáciami, ktoré porušujú tento kódex.

6.11 Uvedomiť si, že porušovanie tohoto kódexu nie je v zhode so správaním sa profesionálneho softvérového inžiniera.

6.12 Vyjadriť znepokojenie ľuďom zúčastneným na závažnom porušení tohoto kódexu, keď to nie je nemožné, kontraproduktívne alebo nebezpečné.

6.13 Oznámiť podstatné porušenia tohoto kódexu príslušným autoritám, keď je jasné, že konzultácie s dotknutými osobami sú nemožné, kontraproduktívne alebo nebezpečné.

Princíp 7: Kolegovia

Softvéroví inžinieri by mali byť spravodliví ku svojím kolegom a mali by ich podporovať. Obzvlášť by softvéroví inžinieri mali:

7.01 Nabádať kolegov aby sa pridržiavali tohto Kódexu.

7.02 Napomáhať kolegom v profesionálnom raste.

7.03 Plne dôverovať práci ostatných a zdržať sa pripisovania nadmerných zásluh.

7.04 Posudzovať prácu iných objektívne, poctivo a náležite zdokumentovaným spôsobom.

7.05 Korektne vypočuť názory, obavy alebo sťažnosti kolegov.

7.06 Napomáhať kolegom k uvedomeniu si súčasných štandardných praktík vrátane bezpečnostnej politiky a procedúr na ochranu hesiel, súborov a ostatných citlivých informácií a bezpečnostné opatrenia všeobecne.

7.07 Nepodnikať žiadne nekorektné kroky v kariére ostatných kolegov, i keby obavy zamestnávateľa, klienta alebo záujem verejnosti nútili softvérových inžinierov, v dobrej viere, zapochybovať o schopnostiach kolegov.

7.08 V situáciách mimo svojho poľa pôsobenia požiadať o pomoc odborníka, ktorý má kompetencie v týchto oblastiach.

Princíp 8: Osobnosť

Softvéroví inžinieri by sa mali zúčastňovať celoživotného vzdelávania týkajúceho sa ich profesnej praxe a mali by presadzovať etický prístup v profesnej praxi. Obzvlášť by sa mali sotvéroví inžinieri usilovať:

8.01 Rozširovať svoje znalosti analýzy, špecifikácie, návrhu, vývoja, údržby a testovania softvéru a súvisiacich dokumentov spolu s manažmentom vývojového procesu.

8.02 Zlepšovať svoju schopnosť vytvárať bezpečný, spoľahlivý a prospešný vysoko kvalitný softvér za rozumnú cenu a v rozumnom čase.

8.03 Zlepšovať svoju schopnosť vytvoriť presnú, informatívnu a dobre napísanú dokumentáciu.

8.04 Zlepšovať svoje porozumenie softvéru a príslušnej dokumentácie, na ktorých pracujú, a prostredia, v ktorom budú použité.

8.05 Zlepšovať svoju znalosť príslušných štandardov a práva spravujúceho softvér a príslušnú dokumentáciu, na ktorých pracujú.

8.06 Zlepšovať svoju znalosť tohto Kódexu, jeho interpretácie a jeho aplikácie v praxi.

8.07 Nezaobchádzať nespravodlivo s nikým z dôvodu akýchkoľvek irelevantných predsudkov.

8.08 Nenabádať iných na vykonanie činov zahŕňajúcich porušenie tohto Kódexu.

8.09 Poznať, že osobné porušenie tohto Kódexu je nezlúčiteľné s postavením softvérového inžiniera.

Etika ako zabezpečenie kvality?

Robert Dvoran

Etický kódex v spojení so svedomím jednotlivca sa často stáva vedúcim impulzom v situáciách, kedy sa rozhoduje o „duši“ softvérového inžiniera. Etický kódex síce inžinierovi nepovie ako má vytvoriť kvalitný produkt, ale povie mu, či sa mu podarilo dosiahnuť úspešne cieľ v podobe dobrého výsledku, alebo nie, čím sa stáva akýmsi univerzálnym nástrojom pre zabezpečenie kvality a udržanie vysokého štandardu profesie softvérový inžinier. Táto esej reaguje na prvý princíp Etického kódexu softvérového inžiniera [1] – Verejnosť.

Predhovor

Etika a softvéroví inžinieri, spojenie pre mnohých exotické, pre iných každodenné. Vytváranie požiadaviek a noriem na správanie jednotlivcov je však vlastné každej ľudskej spoločnosti, a preto by nemalo nikoho prekvapiť, že aj softvéroví inžinieri si vytvorili svoj etický kódex.

V oblastiach, ktoré sa nachádzajú vo fáze vývoja môžeme často pozorovať súčasnú existenciu viacerých alternatív, z ktorých sa postupne vykryštalizuje konečné riešenie. Nie inak je to aj v oblasti etických kódexov, o čom svedčí aj existencia viacerých verzií od rôznych odborných organizácií. Hoci sú v týchto dokumentoch často rozdiely, zhodujú sa v dôležitosti vzťahu softvérových inžinierov k verejnosti. Pravidlá týkajúce sa tohto bodu sa vyskytujú na popredných miestach, čo by však malo byť jasné už z definície pojmu etika. *„Etika je sústava noriem o mravnom, morálnom a spoločensky vhodnom správaní“* [2]. Je teda zrejmé, že zásady správania jednotlivca voči spoločnosti (verejnosti) by mali byť prioritou každého etického kódexu.

Ad 1.01

Schopnosť a ochota prijať plnú zodpovednosť za svoje činy a prenesené do profesionálnej oblasti za svoju prácu je jedným z hlavných predpokladov pre akékoľvek uvažovanie o etickom správaní. Dobrovoľné prijatie zodpovednosti je úzko spojené so slobodnou vôľou a možnosťou voľby. Veď asi nikto nebude považovať vynútené rozhodnutie za „hrdinský“ čin. Väčšina ľudí totiž koná eticky, ale často len v určitých hraniciach, kým im to nepôsobí veľkú bolesť. Nie je totiž veľký problém udržať si čistú vestu, keď nikde nie je blato. Avšak v kritických situáciách zotrvajú len tí najodolnejší a najviac motivovaní.

Napriek skutočnosti, že vývoj softvéru je veľmi hektický a dynamický, sú mnohé dnešné rozhodnutia v tejto oblasti určované rozhodnutiami vykonanými v „dávnej“ minulosti. Príklady ani nemusíme veľmi hľadať, veď množstvo z nás sa s nimi stretáva denne. Ako príklad môže poslúžiť tzv. vírus tisícročia, ktorý hrozil uvrhnúť celý svet do úplného chaosu, alebo problémy s niektorými operačnými systémami a hardvérovými platformami, ktoré kvôli spätnej kompatibilite musia namiesto systematických riešení používať rôzne barličky a obchvaty, čím používateľovi pútajú na nohu olovenú guľu, ktorá bráni v rozlete aj najschopnejším jedincom.

Vo svetle tejto skutočnosti je potreba uváženia každého rozhodnutia a pripravenosť niesť zaň plnú zodpovednosť ešte výraznejšou požiadavkou na integritu a etické správanie softvérových inžinierov. V nedávnej minulosti sme boli svedkami dôsledkov takéhoto rozhodnutia a náklady na zabránenie katastrofe boli nemalé. Hoci si myslím, že vírus tisícročia, bol príliš nafúknutý balón, je aspoň z vypracovaných katastrofických scenárov zrejmé, na čo všetko vplývajú rozhodnutia vykonané v oblasti vývoja softvérových produktov.

Ad 1.02

Softvérové inžinierstvo má v dnešnej dobe potenciálne neobmedzené možnosti zmeniť zabehané mechanizmy v každej oblasti života. Avšak nie každá zmena je znakom napredovania. Preto je úlohou každého softvérového inžiniera na základe svojej význačnosti, teda znalostí z profesionálnej oblasti, napomáhať vývoju správnym smerom, či už podporovaním noviniek s pozitívnym prínosom alebo umierňovaním nevhodných požiadaviek alebo úsilia. Súčasťou tejto úlohy by malo byť aj overenie, že sa rozhodovania zúčastňujú len ozajstní odborníci, ktorí však nezabudnú koho hlavne sa bude ich rozhodnutie týkať, teda na verejnosť.

Ad 1.03

Skutočnosť, že každý program, ktorý má aspoň x riadkov, obsahuje chybu je všeobecne známa. Hoci x sa vytváraním a overovaním štandardných postupov pri tvorbe softvéru zvyšuje, v dnešnej dobe ešte určite nie je také veľké, aby sme mohli

z pôsobnosti tohoto výroku vyňať ktorýkoľvek softvérový projekt, čo je dobre tiež dobre známe a dlhoročnými skúsenosťami množstva vývojových pracovníkov overené. Dúfajme, že raz tento výrok prekonáme. V dnešnej dobe tak stále vrhá neradostné svetlo na schopnosť úspešne sa držať zásady schváliť len produkt, ktorý vyhovuje všetkým požiadavkám. Avšak etický kódex nie je o krutej realite, ale o ideáloch a princípoch. Preto aplikácia tohoto pravidla v praxi bude spočívať vo vynaložení maximálneho úsilia, pretože to je aj tak najviac, čo môžeme použiť, hoci je možné, že ani to nebude stačiť. Pravdepodobný nepriaznivý dopad by aspoň čiastočne mohlo znížiť úplné a jasné prezentovanie všetkých známych nedostatkov spolu s produktom, čo síce určite nezvýši profit, ale aspoň človek môže kľudnejšie spávať. Ten stratený profit nie je jednoduché oželieť, ale asi nikto nečaká, že správať sa eticky je najjednoduchšia z možných ciest.

Nemyslím si však, že prezentovanie všetkých chýb je nejaký zázračný všeliek, ak počet chýb v produkte prekročí určitú hranicu, vytvára to totiž už len horší obraz. Príkladom nech je operačný systém Windows 2000, ktorý podľa interných informácií svojho výrobcu, firmy Microsoft, v čase svojho uvedenia na trh a začiatku predaja obsahoval viac ako 63,000 chýb [3], z ktorých viac ako 25,000 bolo označených za potenciálne vážne problémy. Podobný postup okrem toho, že je zrejme v rozpore s množstvom odstavcov etického kódexu, hraničí až s drzosťou voči používateľovi. Azda si „všemocní“ softvéroví vývojoví pracovníci nemyslia, že potenciálni používatelia na všetko skočia a následne to zoberú s humorom, za ktorý si pekne zaplatili.

Ad 1.04

S uvedeným úzko súvisí aj ďalšie pravidlo, ktoré odporúča, aby softvérový inžinier, ktorý odhalí nejaké chyby ohrozujúce blaho všetkých zúčastnených, tieto prezentoval vhodnej osobe alebo autorite. Toto je pravidlo, ktoré so sebou prináša asi najväčšie riziko osobných strát, ak dôjde ku konfliktu z dôvodu jeho dodržiavania.

Ak totiž usilovný inžinier narazí na manažment, ktorý nie je pripravený prijať jeho opodstatnené pripomienky s náležitým prístupom, ocitne sa v situácii, z ktorej môže byť jediným východiskom, pri zachovaní vlastnej integrity, len odchod z projektu alebo dokonca z organizácie [4]. Možným vyústením problému by mohlo byť rozhodnutie softvérového inžiniera veriť v svoj manažment, že nakoniec dokáže vzniknutú situáciu korektne vyriešiť, a nechať ho konať svoju povinnosť, čo však nie je vždy skutočnosť, a potom ostáva možnosť tvárit sa, že žiaden problém neexistuje, čo nie je síce etické, ale z krátkodobého hľadiska asi najmenej bolestivé. Alebo si bude stáť za svojim za každú cenu, a buď zvíťazí v následnej konfrontácii, čo často býva draho zaplatené víťazstvo, pretože jednotlivec, ktorý takto „rozvráti“ manažment sa zväčša potom nebude môcť zaradiť do pôvodnej pozície, alebo si zvolí už spomínaný odchod.

Navyše si treba uvedomiť, že v niektorých oblastiach nasadenia softvéru je každá objavená chyba potenciálnym alebo dokonca reálnym nebezpečenstvom pre verejnosť. Takouto oblasťou sú napríklad operačné systémy alebo zabezpečovacie systémy. Ak je

takáto „chyba“ vytvorená zámerne, čo je vrcholne neetické, môže nastať situácia ako vo filme Sieť, kde hlavná hrdinka po objavení takýchto zadných dvierok do bezpečnostného systému nasadeného v širokej skupine rôznych úradov je pripravená o vlastnú identitu a minulosť. V dnešnej dobe informačných systémov totiž nie je potrebné vyrábať udalosti, teda napríklad spáchať nejaký trestný čin a zvaliť to na niekoho, stačí ak o tom na vhodných miestach vzniknú záznamy, čo je o mnoho jednoduchšie.

Ad 1.05

Podobne ako by mal každý softvérový inžinier niesť plnú zodpovednosť za svoju prácu, mal by cítiť aj zodpovednosť voči svojej profesii. Preto je zrejmé, že je v záujme všeobecného blaha, aby softvéroví inžinieri spolupracovali pri riešení problémov s vážnym spoločenským dopadom, ktoré sú spôsobené softvérom. Veď čo sa týka softvérového inžinierstva, týka sa aj každého softvérového inžiniera. Takisto sa často stáva, že jedna čierna ovca poškodí celé stádo.

Ad 1.06

Ak by sa malo striktne dodržiavať odporúčanie o nehovorení neprávd a polopravd, bolo by asi nutné zrušiť reklamné a marketingové oddelenia všetkých firiem, čo asi nie je možné. Preto by aspoň každý inžinier mal uvádzať vždy len pravdivé informácie, pretože stratená dôvera sa nadobúda veľmi ťažko a každá nepravda o softvéri znižuje dôveryhodnosť celej profesie. Okrem toho, uvádzanie polopravd len zbytočne mystifikuje softvérové inžinierstvo a posúva ho do role pavedy, ktorej výsledky fungujú len na heslo, a to tiež nie vždy.

Ad 1.07

Všeobecné zjednocovanie a s ním spojené prekonávanie rôznych predsudkov a iných skutočností, ktoré môžu znižovať kvalitu života rôznych skupín ľudí, je momentálne veľmi aktuálnym trendom, avšak nie módou, pretože vyplýva z vlastnej podstaty etického správania, preukazovania určitej úcty všetkým zložkám spoločnosti. Je len prirodzené, aby aj softvéroví inžinieri prispeli svojou troškou do mlyna a pokúsili sa prekonať všetky prekážky spôsobené rozdielnosťou, vo všetkých svojich podobách, aby výhody, ktoré softvérové produkty prinášajú, boli využiteľné všetkými, ktorí o to stoja, a to bez výnimky.

Ad 1.08

Je želateľné, aby každý softvérový inžinier poskytol svoje znalosti na dobré účely a zvyšovanie všeobecnej úrovne znalostí z oblasti. Nejde len o to, aby mal kto našu prácu oceniť, veď keď tomu nikto nerozumie, tak ako by to nebolo, ale aj o možnosti ďalšieho napredovania. Ťažko sa totiž stavia dom tam, kde musíte zakaždým postaviť základy. Okrem toho, všetci predsa žijeme na tej istej Zemi, a preto ak ju chceme zmeniť k lepšiemu, mal by každý pomôcť tým, čo vie najlepšie. Zmienky o príhodách potvrdzujúcich mysteriózny a mystifikovaný status softvérového inžinierstva a softvéru všeobecne môžeme nájsť v rôznej literatúre [5].

Záver

V histórii sa často ukázalo, že skupina jedincov spojená spoločnými pevne stanovenými a všetkými rešpektovanými pravidlami dokázala prekonať širokú škálu prekážok a pritom zostala stáť pevne na svojich nohách a neupustila od svojich zásad. Kde totiž zlyháva rozum, často pomáha viera v princípy. Tak ako zásady „klasickej“ etiky nás vedú, aby sme boli lepší ľudia, tak aj etický kódex softvérového inžinierstva nám, dúfajme, pomôže, aby sme sa stali lepšími softvérovými inžiniermi.

Dovoľte, aby som svoju esej ukončil slovami prebranými od fiktívnej osoby, Dr. Harryho Yordera, profesora informatiky s teologickým vzdelaním [6]: „Prišiel som k záveru, že prelínanie sa hraníc medzi etickými, technickými a legálnymi otázkami je spôsobené skutočnosťou, že softvérové inžinierstvo je ešte stále v detskom veku. Etické otázky sú také významné, pretože neexistujú technické a legálne vzory pre riešenie daných problémov. Hlavne vo fáze vývoja a testovania neexistujú žiadne štandardné postupy. Existujú síce štandardy, ale nie sú to ozajstné štandardy. Medzi informatikmi často koluje vtip, že jediná dobrá vec na štandardoch je to, že ich tak veľa, že si človek môže vyberať.“

Použitá literatúra:

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

[2] Ivanová-Šalingová, M., Maníková, Z. (1983). *Slovník cudzích slov.* SPN, 1983, Bratislava.

[3] Foley, M.J. (2000). Can Microsoft squash 63,000 bugs in Win2K. *PC Week News*, February 11, 2000, WWW: http://www.zdnet.com/pcweek/stories/news/0,4153,2436920,00.html

[4] Koopman, P.(1999). On Being the Bearer of bad News, http://onlineethics.org/essays/bad-news1.html

[5] McCarthy, J. (1999). Softwarové projekty. Microsoft Press, 1999, Praha.

[6] Epstein, R.G. (1998). The ‘Killer Robot‘ Story. CWRU 1998, http://onlineethics.org/cases/robot/robot.html

Ja, on a on

Peter Jehlár

Úvaha nad vzťahmi softvérového inžiniera a zamestnávateľa alebo zákazníka. Zamyslenie sa nad jednotlivými bodmi druhého princípu „Zákazník a zamestnávateľ" Etického kódexu softvérového inžiniera (verzia 5.2), vytvoreného spoločnou komisiou IEEE-CS a ACM [1], z niektorých uhlov pohľadu.

Predstavenie

Dovoľte aby som predstavil tri osoby z nadpisu. Prvá osoba – „ja" predstavuje softvérového inžiniera. Tretia osoba – „on" je zamestnávateľ pre ktorého „ja" pracuje a prostredníctvom ktorého uspokojuje potreby druhej osoby – „on" – zákazníka. Tieto tri osoby vystupujú aj v druhom princípe Etického kódexu softvérového inžiniera. Deväť bodov druhého princípu na prvý pohľad vyzerá jednoducho, jednoznačne a ľahko realizovateľne. Je to skutočne tak?

Osoby zákazníka a zamestnávateľa možno generalizovať na jednu, vzhľadom na vzťah k softvérovému inžinierovi, ktorý je k zákazníkovi aj zamestnávateľovi takmer rovnaký, líši sa väčšinou malými rozdielmi.

Ukážte svoju pravú tvár

Prvý bod (2.01) hovorí o poskytovaní služieb softvérového inžiniera v oblastiach jeho pôsobnosti. Má robiť to, čo vie najlepšie, s čím má skúsenosti, to čo vyštudoval, nemal by byť „capom – záhradníkom". Tento bod je všeobecný, netýka sa len profesie softvérového inžiniera ale aj ostatných profesií.

Ďalej sa v prvom bode hovorí o čestnosti a otvorenosti softvérového inžiniera pri informovaní zákazníka a zamestnávateľa o svojich obmedzeniach a nedostatkoch. Každý s nás chce byť čestný a otvorený, no väčšinou to tak nie je. Takmer vždy niečo prikrášlime, aby sme v očiach druhých vyzerali čo najlepšie. Väčšinou sa nám to „darí", aspoň si to my myslíme. No v skutočnosti, čím dlhšie sa „ospevujeme", tým

horšie vyzeráme v očiach druhej strany. Skôr či neskôr nás na základe našich skutkov odhalí, ak nás neodhalila hneď na začiatku. Druhá strana používa také isté alebo podobné metódy ako my, preto väčšinou vie, kedy nehovoríme celkom pravdu, alebo hovoríme „pravdu“, ktorá je až na neuverenie.

Jedným z príkladov kedy chceme zapôsobiť je prijímací pohovor, napríklad do nového zamestnania. Na pohovore hovoríme pravdivo, ale keď sa dá, niektoré „detaily“ prikrášlime. Veď prečo nie, okrem nás je tu ešte ďalších desať ľudí. Vedomostne aj skúsenosťou sme na tom lepšie ako konkurencia. Tak prečo si ešte trochu nevylepšiť profil, keby náhodou… „Náhodou“ sa stane, že nás neprijmú. Prečo? Ako nás odhalili? Pri našom pohovore sme chceli vyzerať lepšie, tvárili sme sa ako niekto druhý. Pozabudli sme pritom na to, že sme neboli sami sebou, neboli sme otvorení, „nevyložili sme karty na stôl“. Rozprávali sme tak, ako sme si mysleli, že máme rozprávať [2]. Skrývali sme sa za masku. Ale kto by už len chcel kúpiť mačku vo vreci? Tento záver som experimentálne overil v praxi pri prijímaní do niekoľkých zamestnaní.

Na druhej stane, prečo by sme mali hovoriť len čisté fakty a povedať o sebe radšej menej a nespomenúť pútavou, i keď „vhodne upravenou“, formou všetky svoje prednosti? Veď ostatní na prijímacom pohovore o sebe povedia veľmi veľa vecí v tom najlepšom svetle. Budúci zamestnávateľ si radšej vyberie takú osobu, o ktorej síce vie, že nie všetko čo povedala je pravda, ale „opticky“ pôsobí lepším dojmom. Veď ak sa z nej vykľuje niekto iný, môže napokon dotyčnú osobu prepustiť. Zamestnávateľ si nás určite nevyberie pokiaľ budeme vyzerať veľmi skromne a svoje úspechy zamlčíme, čím budeme pôsobiť dojmom priemerného softvérového inžiniera. Budúci zamestnávateľ hľadá priebojných ľudí plných elánu ochotných sa popasovať aj s neriešiteľnými situáciami. Nehľadá predsa osobu na voľné miesto upratovačky, ale na pozíciu softvérového inžiniera.

Za otázku stojí, či máme čestne a otvorene informovať aj o svojich obmedzeniach, ktoré napríklad spôsobili škodu predchádzajúcemu zamestnávateľovi, resp. zákazníkovi. Obávam sa, že takto by sme si nového zamestnávateľa nenašli.

Čo teda o sebe prezradiť? Takmer celú pravdu. Môžeme ju trochu prikrášliť. Treba mať na pamäti staré príslovie: všetkého s mierou.

Okrádaním druhého okrádate sami seba

V bode dva (2.02) sa hovorí o nepoužívaní softvéru, ktorý je získaný alebo ponechaný nelegálne alebo neeticky. Oficiálne sa k myšlienke nepoužívať nelegálny (pirátsky) softvér hlási takmer každý. No v skutočnosti veľmi veľa ľudí má na svojom počítači softvér, ktorý je nelegálny. Môže ísť o operačný systém, kancelársky balík, vývojové prostredie, kresliaci program, jednoúčelovo zameraný program alebo malý pomocný program – „utilitku“… Každý s vyššie spomenutého softvéru alebo aj iný je možné získať zadarmo aj používať zadarmo. Pokiaľ používame takýto softvér je všetko v poriadku.

Iná situácia nastáva ak používame softvér, za ktorý sme mali zaplatiť, ale nezaplatili sme. Na prvý pohľad nám to prináša výhody, no v skutočnosti to tak nie je. Výhodné je to možno z ekonomického hľadiska pre tých, ktorí používajú softvér vyrobený softvérovými inžiniermi. Zväčša majú „inštalačky“ vypálené na CD alebo nahraté na pevnom disku. A to je všetko. Tieto osoby ale prichádzajú o technickú podporu, zľavy pri nákupoch ďalších verzií,… …a tí svedomitejší možno aj o pokojný spánok (aj keď nemajú legálny softvér, majú aspoň svedomie). So softvérovými inžiniermi, oproti bežným používateľom, to ale nie je podobné, je to oveľa horšie! Softvéroví inžinieri, ktorí používajú softvér nelegálne sú sami proti sebe! Prečo?

Gazda, ktorý pred dažďom nepozhŕňa a neskryje seno, ktoré sa mu suší pre kravu, len preto, že je lenivý, môže na to doplatiť. Môže pršať aj niekoľko dní a seno mu zhnije. Pokiaľ má sena málo, v zime sa mu to vypomstí. Zdochne mu krava, ktorá nemá čo žrať. Síce uspokojil svoje momentálne „potreby“, ale zabudol na budúcnosť.

Podobne je to aj so softvérovým inžinierom. Ak používa nelegálny softvér alebo tomu napomáha ubližuje nepriamo sám sebe. Za softvér, ktorý používa, výrobcovi nezaplatil. Náklady výrobcu sa odrážajú v cene softvéru. Preto ak výrobca predá málo kusov svojho produktu, cena jedného kusu softvéru bude vyššia ako keby ich predal viac. Pri vysokej cene to odradí určité percento potenciálnych zákazníkov. Výrobca sa snaží znížiť svoje náklady na minimum aby cena softvéru bola čo najnižšia a predalo sa ho čo najviac. Náklady obmedzí výrobou softvéru s menšou funkčnosťou, prípadne kvalitou. Na tvorbu softvéru s menšou funkčnosťou treba aj menej softvérových inžinierov. Tým pádom klesá dopyt zo strany výrobcov po softvérových inžinieroch, čo má za následok nižšiu trhovú hodnotu softvérového inžiniera. Nie ste to práve vy, ktorému šéf odmietol „zatiaľ“ zvýšiť plat? [3]

Môžete namietať, že aj v súčasnosti je nedostatok softvérových inžinierov. Tak prečo mať legálny softvér? Jedným s ďalších dôvodov je porušovanie zákona. Softvér sa z právneho hľadiska, nielen u nás, ale aj vo svete, považuje za nehmotný majetok – autorské dielo. Nositeľmi práv týkajúcich sa softvéru sú jeho autori, resp. firma, v ktorej bol vytvorený. Na akékoľvek disponovanie so softvérom – autorským dielom je potrebný súhlas nositeľa autorských práv, pričom tento súhlas sa udeľuje na základe zmluvy nazývanej licenčná dohoda. Táto dohoda je priložená k softvérovému produktu alebo sa zobrazí na úvodnej obrazovke po spustení softvéru.

Jednou z foriem nedodržiavania zákona je aj distribúcia nelegálneho softvéru. Ako príklad uvádzam prípad odsúdeného piráta na Slovensku. 14. januára 1998 vyniesol súd vôbec prvý rozsudok nad pirátskym obchodníkom na Slovensku. Obžalovaný bol odsúdený na 8-mesačný podmienečný trest odňatia slobody. Zároveň mu prepadla zásoba CD-ROM médií a dva osobné počítače. Obchodník vyrábal a distribuoval pirátsky softvér spoločností Autodesk, Microsoft a Novell, z čoho dosiahol zisk vo výške 11 413 USD (400 000 Sk). [4]

Štatistiky uvádzajú, že na Slovensku v roku 1998 bol 50% podiel nelegálneho softvéru, čo predstavuje straty v cene 11,2 milióna USD (400 miliónov Sk). Oproti roku 1994 to predstavuje 16% pokles. V celosvetovom meradle to v roku 1998 bol 38% podiel, straty predstavovali 11,4 miliardy USD. Ako bolo spomenuté vyššie, množstvo nelegálneho softvéru ovplyvňuje aj množstvo pracovných miest. Viac o predpovediach,

vývoja počtu pracovných miest vzhľadom k množstvu nelegálneho softvéru a iné štatistiky môžete nájsť aj v [5].

Svojvoľné využívanie cudzieho majetku

V treťom bode (2.03) sa hovorí o používaní majetku zákazníka alebo zamestnávateľa. Majetok sa má používať len spôsobmi náležite oprávnenými a s vedomím a so súhlasom zákazníka alebo zamestnávateľa.

Dosť často sa v našich podmienkach stáva, že zamestnanci si zoberú inštalačné médiá bez vedomia zamestnávateľa domov a nainštalujú si doma softvér, ktorý používajú v zamestnaní. Neuvedomujú si alebo neberú ohľad na to (ako bolo spomenuté vyššie), že okrem iného, používajú doma softvér nelegálne.

Iný uhol pohľadu je ten, že softvér (za predpokladu, že zamestnávateľ má daný softvér legálne), ktorý si zamestnanec doma nainštaloval, v podstate ukradol svojmu zamestnávateľovi. Zamestnávateľ ho kúpil pre svoje potreby a zamestnanec ho môže použiť na súkromné ciele.

Ako majitelia stavebnej firmy by sme neboli veľmi nadšení, ak by si náš murár bez nášho vedomia a súhlasu „požičal“ nákladné auto a vozil materiál na stavbu svojho rodinného domu. V tomto prípade by sme vedeli murára ľahko odhaliť. Horšie je to so softvérom, ktorý je nehmatateľný a tým pádom sa neopotrebúva. Preto nám z pozície zamestnávateľa ostáva dôsledne kontrolovať pohyb inštalačných médií.

Dôverujte, ale preverenému

Vo štvrtom bode (2.04) sa hovorí, že softvérový inžinier má v prípade potreby zabezpečiť, aby dokument, ktorému prikladá zamestnávateľ alebo zákazník určitú váhu, bol schválený.

Softvérový inžinier nemôže vychádzať z nejakého dokumentu, o ktorého pôvode a správnosti nevie nič. Pokiaľ by použil neschválený dokument, pravdepodobne by od tej doby všetko jeho vynakladané úsilie vyšlo na zmar. Dokument by s najväčšou pravdepodobnosťou nebol v súlade s prebiehajúcim postupom prác na projekte. Jeho použite by neviedlo k úspešnému napredovaniu v projekte a jeho úspešnému ukončeniu.

Nevynášajte recepty z kuchyne dôverčivého kuchára

Piaty bod (2.05) pojednáva o udržiavaní dôverných informácií v tajnosti, ktoré softvérový inžinier získal počas svojej odbornej pôsobnosti, ale iba vtedy, ak utajovanie týchto informácií je v zhode s verejným záujmom a v zhode so zákonom.

Profesia softvérového inžiniera je jedna z mála profesií, ak nie jediná, ktorej sa podarí získať veľké množstvo dôverných a citlivých informácií. Aby softvérový inžinier mohol zrealizovať projekt, ktorého výsledkom je softvér pre konkrétneho zákazníka, musí sa dôsledne a do detailov oboznámiť s danou problémovou doménou zákazníka. Často vytvára systémy „šité na mieru“, v ktorých je zohľadnený súbor poznatkov a skúseností („know-how“) zákazníka. Zákazníkovi umožňujú prežiť v tvrdom konkurenčnom prostredí. Zákazníkovi neostáva nič iné len sa spoľahnúť na serióznosť firmy. Softvérový inžinier pri tvorbe a údržbe zákazníckeho softvéru prichádza aj k veľkým množstvám údajov v databázach. Práve zneužitie údajov je veľmi citlivá záležitosť. Väčšinou dochádza pri ich zneužití k strate dôvery voči firme, ktorá ich vlastní – zákazníkovi, ale aj voči firme, ktorá sa stará o použiteľnosť týchto dát – zamestnávateľa softvérového inžiniera.

Pre softvérového inžiniera môže byť lákavá predstava poskytnutia dôverných informácií konkurencii zákazníka alebo zamestnávateľa, ktorá mu môže náhle zvýšiť životnú úroveň. Je to ale veľmi nebezpečné, pretože pôvodca úniku dôverných informácií sa väčšinou odhalí a je podľa zákona stíhaný.

Môže sa stať, že vás vychýrený kuchár pustí do svojej kuchyne. Kuchár vám dôveruje a ukáže vám ako robí svoju špecialitu. Necháte sa zlákať, a prezradíte recept inému závistlivému kuchárovi, čím si získate jeho náklonnosť, ale len naoko. Stratíte dôveru starého dôverčivého kuchára, pretože jeho špecialitu ovláda aj niekto iný, stratil konkurenčnú výhodu. Vaša zlá povesť sa rýchlo rozšíri medzi všetkými kuchármi. Začnú sa vás báť. Báť sa vás bude aj závistlivý kuchár, pretože pozná váš charakter. Kde má záruku, že sa k nemu nezachováte podobne? Komunita kuchárov sa vám začne vyhýbať.

Ako softvérový inžinier získate dôveru zamestnávateľa a riešite projekt pre zákazníka. Zákazník vášho zamestnávateľa tiež považuje za dôveryhodnú osobu, tým pádom aj vás. Necháte sa zlákať konkurenciou zákazníka a poskytnete jej dôverné informácie. Pri odhalení strácate dôveru svojho zamestnávateľa. Stratili ste ale dôveru aj vašich potenciálnych zamestnávateľov. Kde majú záruku, že sa k nim nezachováte podobne?

Tým, že si to uvedomuje, je za to zodpovedný

Doteraz bolo spomínané čo by mal alebo nemal softvérový inžinier, ak sa ho to bezprostredne týka.

Body šesť a sedem (2.06 a 2.07) hovoria o tom, že softvérový inžinier má rozpoznávať, zaznamenávať a oznámiť zamestnávateľovi, prípadne zákazníkovi, ak dospeje k názoru, že postup prác na projekte vedie k jeho neúspechu. Tak isto má postupovať, ak všetko podľa neho nasvedčuje tomu, že projekt je drahý, alebo porušuje duševné vlastníctvo. Takisto, ako v prípade projektu, má postupovať aj v prípade softvéru a s ním spojenej dokumentácie.

Lojálnosť nie za každú cenu

Podľa ôsmeho bodu (2.08) nemá softvérový inžinier akceptovať vonkajší nepriaznivý vplyv na prácu pre svojho zamestnávateľa. Podľa tohto bodu sa požaduje oprávnene lojálnosť voči zamestnávateľovi. Zamestnávateľ sa tiež voči svojim zamestnancom (väčšinou) stavia lojálne. Napríklad v prípade neúspechu projektu nesie navonok zodpovednosť on voči zákazníkovi. Preto nie je dobré ak zamestnanec, hlavne v prípade ťažkých chvíľ, nepodporí zamestnávateľa.

Naproti tomu, sa v deviatom bode hovorí o negatívnej zainteresovanosti softvérového inžiniera voči zamestnávateľovi alebo zákazníkovi. Činnosť softvérového inžiniera má byť zameraná proti druhej strane len ak je to v zhode s vyšším etickým záujmom. Je potrebné aby v takomto prípade o svojej činnosti informoval vyššiu autoritu alebo tretiu stranu.

Záver

Druhý princíp Etického kódexu softvérového inžiniera pojednáva o vzťahu softvérového inžiniera a zákazníka, resp. zamestnávateľa. Etický kódex softvérového inžiniera je profesijným etickým kódexom. Prezentuje skupinové, kolektívne etické normy a princípy, ktoré napomáhajú regulovať a usmerňovať činnosť každého jednotlivca – softvérového inžiniera, v súlade so skupinovými etickými normami. Pomáha korigovať vzťah medzi profesiou a spoločnosťou. Má slúžiť ako garancia, že profesia softvérového inžiniera ako celok bude poskytovať vysoko odborné služby a nebude zavádzať druhú stranu – laickú verejnosť, resp. zneužívať jej nevedomosť alebo bezmocnosť v danej situácii.

Použitá literatúra

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

[2] Carnegie, D.: Ako sa zbaviť starostí a začať žiť. PRÍRODA. Bratislava 1998.

[3] Andrášik, L.: Prednášky z predmetu: Teória modelovania ekonomických systémov. Fakulta elektrotechniky a informatiky STU Bratislava. Bratislava 1999.

[4] Prvý odsúdený pirát na Slovensku. BSA SK, 1998. http://www.bsa.sk.

[5] 1998 Global Software Piracy Report. (A study conducted by International Planning and Research Corporation For the Business Software Alliance), 1998. http://www.bsa.org.

Potulky životom alebo s čím sa v živote stretávame

Matej Černák

Esej je zameraná na ukážku použitia pravidiel definovaných druhým princípom etického kódexu softvérového inžiniera [1]. Na príbehoch zo života poukazuje na polemiku správania a na potrebu ukázania správnej cesty v kritických situáciách.

Predhovor

Cieľom tohto článku je na vymyslenom, no zo skúseností motivovanom príbehu, poukázať na opodstatnenosť časti Zákazník a zamestnávateľ v etickom kódexe. Hlavný hrdina svojím počínaním vyjadruje postoje autora v jednotlivých situáciách. Riešenie danej situácie je potom vysvetlené ako aj otázkami kladenými hlavným hrdinom, tak aj výsledným správaním načrtnutým v úvode ďalšej kapitoly. Postup a opis problematiky je potom prirovnaný a vysvetlený konkrétnym pravidlom etického kódexu.

Úvod

Naše rozprávanie sa vracia do čias keď hlavný hrdina rozprávania, Martin, začal objavovať čaro abstraktného myslenia. Už na strednej škole sa aktívne zúčastňoval rôznych súťaží, kde podľa pravidiel zápasil o úspech so svojimi rovesníkmi. Dosahoval celkom slušné výsledky, a tak ho starosti bežného života zdanlivo obchádzali.

Všetko bolo v najlepšom poriadku až raz. Prišlo to úplne nečakane. Martinov starší brat zomrel na následky autonehody, ktorá bola zavinená opitým vodičom. Ohromilo to celú rodinu a Martin bol z toho dosť deprimovaný. Stále si kládol ťažké otázky: Prečo sa to muselo stať zrovna jeho bratovi? Ako ďalej? Ako je možné riadiť automobil pod vplyvom alkoholu? Či iné. Tieto trpké myšlienky ho priviedli k idei, zamyslieť sa nad všetkým, čo ho v živote stretne a v duchu hesla: "*Všetko zlé je na niečo dobré*" si zobrať z toho ponaučenie. Tohto predsavzatie sa držal aj počas štúdia na univerzite, kde promoval s vyznamenaním. Získané vedomosti v oblasti softérového inžinierstva

plánoval využiť aj vo svojom budúcom zamestnaní, a keďže patril k najúspešnejším žiakom ročníka, očakával, že sa oňho budú zamestnávatelia biť.

Prvé skúsenosti

Aké bolo však jeho prekvapenie, keď z konkurzov odchádzal so sklonenou hlavou. Miesta, o ktoré mal záujem mu zobrali spolužiaci, ktorí vedomostne ďaleko za ním zaostávali. Príčina? Nebol proste schopný zavádzať budúcich zamestnávateľov svojimi skúsenosťami a schopnosťami. Aj keď jeho schopnosti a vedomosti by vybranej práci plne vyhovovali, prednosť dostali uchádzači, ktorí ako tvrdili majú v danej oblasti dobré skúsenosti.

Tu sa Martin zarazil a po niekoľkých neúspechoch začal nad problémom uvažovať. Je správne preceňovať svoje schopnosti, dúfajúc, že na to zamestnávateľ nepríde? Považuje sa uvedenie nie úplne presnej, či zavádzajúcej informácie za neetické? Mal by sa aj on prispôsobiť okoliu a začať používať rovnaký postup? Prečo zamestnávatelia nevedia odhaliť tento trik a nevypracujú si primeraný výber zamestnancov?

Tento príbeh poukazuje na pravidlo 2.01, ktoré apeluje na čestnosť a otvorenosť softvérového inžiniera.

Začiatky

Po prvých neúspechoch však prišlo dlho očakávané riešenie. Martin nastúpil na svoje prvé pracovisko, a tým sa jeho problém vyriešil.

Jeho úlohou bolo dohliadať na systémy a zariadenia firmy. V rámci svojej činnosti spolupracoval aj s istou opravárenskou spoločnosťou, ktorá im zabezpečovala servis techniky. Nikdy nevedel pochopiť, ako je možné, že sa zariadenia z opravy vracali tak zaprášené. Prístroje fungovali, takže so sťažnosťami u predstavenstva nepochodil. Príčinou neprimeraného stavu zariadení bol prístup zo strany dodávateľa, ktorý zariadenia skladoval na neprimeranom mieste a nepristupoval k nim s náležitou opaterou. Martinovi bol tento prístup nepochopiteľný. Ako je možné, že firma, ktorá poskytuje takéto služby sa takto správa k zvereným zariadeniam? Ani sa nestihol poriadne zamyslieť a už počul, ako jeho kolegyňa hromží a neúnavne búcha do klávesnice. Vedľa kolega opäť vylial kávu na pracovnom stole a ako upratovačka znova zaliala serverovú miestnosť vedrom vody. Heslo čo je mokré, to je čisté, tu firmu stojí zopár tisíc korún.

Je možné vyžadovať od externých pracovníkov, či dodávateľov korektné správanie ak vlastní zamestnanci nedodržiavajú základné predpisy?

Určite si čitateľ všimol, že som v tomto príbehu nedodržal poradie princípov a príhoda, ktorá sa stala hlavnému hrdinovi sa viaže k bodu 2.03

Bezpečnosť

Ako čas plynul, tak sa aj firma rozrastala a Martin ako šikovný, mladý inžinier postupoval. V dobe, keď mal na starosti bezpečnosť systému musel riešiť nasledujúci problém. Ako vedúci oddelenia zodpovedal za skupinu mladých systémových pracovníkov. Ich obľúbeným koníčkom bolo modifikovať firemné dokumenty. Aj keď tieto boli zabezpečené istým stupňom ochrany, žiadny nebol pre nich nedobytným. Problém nadobudol obrovské rozmery, keď na účtovnom oddelení zistili, že niektorý z ich dokumentov bol modifikovaný neautorizovanou osobou.

Martin stál pred zložitým problémom. Časť problému vyriešilo predstavenstvo, ktoré okamžite ukončilo pracovnú zmluvu s narušiteľmi systému, no na Martinovi ostalo doriešiť zvyšné opatrenia, aby sa tento prípad nemohol opakovať.

Čo viedlo pracovníkov k takémuto spôsobu zábavy? Uvedomovali si dôsledky svojho konania? Mohlo niečo zabrániť takémuto správaniu?

Bezpečnosť je háklivým bodov vo všetkých oblastiach života a aj preto sa tomu venuje aj etický kódex bodom 2.04.

Súboj titanov

Ako už bolo spomínané Martinovo oddelenie zodpovedalo za bezpečnosť systému. Táto činnosť bolo o to dôležitejšia, o čo častejšie boli útoky na systémové informácie. Martin nechápal príčinu, ktorá vedie narušiteľov k svojvoľnému prelomeniu ochrany. Keďže sa v tejto oblasti pohyboval už dlhšiu dobu, poznal rôzne spôsoby ako je možné systém narušiť, ale nikdy ho nenapadlo niečo také uskutočniť.

Mienka spoločnosti je jasná. Nemá sa to. Každý sa tomu bráni, avšak ako Martin zistil 90% z jeho zamestnancov používali v súkromí systémy licencované na ich firmu, či inak získaný softvér.

Je možné aby pracovník zabezpečujúci bezpečnosť jednej firmy, neoprávnene používal produkty firmy druhej, čím ju vlastne poškodzuje? Prečo spoločnosť toleruje túto formu zneužívania?

Martinovi kolegovia mu hneď poskytli množstvo vysvetlení, ktoré však nemali nič spoločné s etickými, či právnymi normami. Ich jediné opodstatnenie spočívali v ospravedlnení správania sa. Ako presvedčiť kolegov, že ich správanie je totožné so správaním narušiteľov, proti ktorým bojujú?

Softvérové pirátstvo nie je verejnosťou považované za zločin veľkého rozsahu a významu. Úlohou bodu 2.02 je zaujať jednoznačné postavenie k tomuto problému, ktorý sa rieši a v budúcnosti sa bude riešiť s ešte väčším rozsahom a dôsledkami.

Prvé rozpory

Postupom času firma stiahla Martina so sekcie bezpečnosti a rozhodla sa využiť jeho analytické schopnosti, skúsenosti s prácou s ľuďmi a poverila ho vedením nového projektu.

Martin bol spočiatku nadmieru spokojný. Robil prácu čo ho bavila, spolupráca v tíme bola na dobrej úrovni a práca im išla od ruky. Celý projekt bol postavený na presnej špecifikácii dodanej zadávateľom. Všetko bežalo podľa plánu do okamžiku, keď Martinova skupina odhalila nedostatky v zadaní, ktoré systém obmedzovali do takej miery, že po jeho zavedení do praxe a naplnením požadovanými údajmi bude prakticky nemožné systém používať.

Martin okamžite podal správu svojim nadriadeným, no obratom mu prišiel dokument, v ktorom bolo jasne povedané, že požiadavky na systém zadali odberatelia a tým aj nesú zodpovednosť za výsledný systém. Následné úpravy, či modifikácie budú prerokované po ukončení projektu a vyúčtované samostatne.

Je možné, aby sa jeho firma takto správala k svojim zákazníkom? Mohol urobiť niečo viac, čo by zabránilo neskorším konfliktom so zadávateľom? Použitá bola zásada 2.07.

Rozpory narastajú

Po nezhodách v predchádzajúcom projekte bol Martin preradený k už rozbehnutému projektu. Na jednej strane bol spokojnejší, že nenesie zodpovednosť za nezdarený projekt, no na druhej strane ho postoj firmy prekvapil.

Z úvah ho prerušila akcia roka. Na stretnutie so spolužiakmi sa veľmi tešil. Keďže Martin patril k spoločensky, obľúbeným hneď sa pustil so spolužiakmi do reči. Po obvyklých frázach od rodiny a detí, prešla reč postupne k pracovnému zaradeniu.

Pri jednom z takýchto rozhovorov sa Martin dozvedel o novom projekte konkurenčnej firmy, ktorý sa práve začal predávať. Ani nevedel ako k tomu prišlo, no dôsledky boli obrovské. Tento projekt bol zameraný na tú istú cieľovú oblasť, na ktorej pracovala Martinova skupina. Avšak výsledné náklady, ktoré dosiahli u konkurencii boli polovičné pri porovnateľnej kvalite produktu.

Martin svoje zistenie prezentoval na pondelňajšom sedení firmy, avšak opäť bol jeho názor ignorovaný. Martinov predpoklad bol správny a aj tento projekt skončil neúspechom. Martin si zbytočne kládol otázky: Čo viac mohol urobiť pre záchranu projektu? Nebolo by lepšie, keby si získanú informáciu ponechal sám pre seba? Má vôbec zmysel byť iniciatívny? Martinovo správanie odpovedalo princípu 2.06.

Posledná kvapka

Posledné neúspechy sa odrazili na Martinovej koncentrácii. Ako sa hovorí, keď sa niečo môže pokaziť, pokazí sa to v najhoršom okamžiku. Tak asi vyzerala aj Martinova

situácia. Okrem rozporuplných projektov vo firme, sa k tomu pridali problémy v rodine. Atmosféra v pracovnom tíme bola dusivá a pracovné nasadenie sa blížilo nule.

Martin si uvedomoval, že postavenie, v ktorom sa nachádza mu neumožňuje venovať sa práci tak ako by si zamestnávatelia predstavovali. Otázkou preňho zostávalo korektné vyriešenie tohto problému. Má vypustiť prácu úplne a ukázať sa tam z času na čas, veď nejaké zásluhy vo firme má alebo si nájsť niečo lepšie?

Martin sa rozhodol pre druhý variant a vybral si dovolenku, ktorá mu mala poskytnúť čas na premýšľanie a súčasne doriešenie si rodinných problémov. Dôležité v jeho rozhodovaní bolo uvedomenie si týchto vonkajších aspektov a riešenie postupne jeden po druhom.

Problematika opísaná postavením hlavného hrdinu opisuje možné škodlivé vplyvy vonkajšieho prostredia. Týmto a ešte aj problematikou vedľajšej pracovnej činnosti sa zaoberá bod 2.08.

Normalizácia

Po dovolenke sa zdanlivo všetko vrátilo do normálu a Martin opätovne začal pracovať na novom projekte. Projekt sa mu páčil a aj práca v tíme napredovala dobrým tempom.

Všetko bolo na najlepšej ceste k ukončeniu projektu. Do konca chýbalo niekoľko dní, keď niektoré z testov naznačovali zlyhanie systému pri špeciálnych prípadoch, ktorých výskyt bol nepatrný, no dôsledky boli katastrofálne. Hrozilo zrútenie celého systému. Martin prezentoval zistený problém vedeniu, avšak to si už ďalší neúspech nedokázalo predstaviť, a tak nariadilo vynechať spomínané testy pri finálnom overovaní. Martinove argumenty, že celý problém môžu vyriešiť za mesiac vedenie neakceptovalo a trvalo na uvoľnení produktu v stanovenom termíne

Martin stál pred dilemou – poslúchnuť vedenie a spoľahnúť sa na malú pravdepodobnosť chyby, alebo zverejniť problematiku verejnosti a tým donútiť firmu k prehodnoteniu svojho postoja. Martin si spomenul na nehodu svojho brata a bolo rozhodnuté.

A práve na tomto prípade je ukázané, že aj keď softvérový inžinier je viazaný zodpovednosťou k zamestnávateľovi, je táto zodpovednosť až po zodpovednosti k okoliu. Práve v tomto prípade prišlo k porušeniu etického princípu, ktorý je opísaný v bode 2.09.

Nové začiatky

Po prepuknutí škandálu bola firma nútená prehodnotiť expedovanie produktu. Martin sa rozhodol ukončiť svoje pôsobenie v spoločnosti a hľadal si nové uplatnenie.

Keďže jeho skúsenosti boli bohaté, nebol problém si nájsť zamestnanie. Zhodou okolností sa mu vyskytla príležitosť pracovať u konkurenčnej firme, kde pracoval aj

jeho spolužiak. Jeho skúsenosti chcela firma hneď uplatniť vo svojich projektoch a vyzvala ho k vypracovaniu dokumentu o spracovávaných projektoch jeho pôvodnej firmy.

Martin, aj keď bol rozčarovaný z posledného pôsobenia vo firme, nechcel prezradiť dôverné informácie, ktoré počas svojho pôsobenia získal. Bol ochotný využiť technológie a vedomosti, ktoré mu boli vlastné, ale predanie informácii o bývalej firme sa mu zdalo neetické. Je pravda, že mohol pomôcť svojmu novému chlebodarcovi, ale za akú cenu? Bolo jeho zamestnanie ozaj z dôvodu jeho schopností, alebo len z dôvodu predbehnutia, či odhalenia konkurencie?

Porušenie zodpovednosti v predchádzajúcej príhode bola opodstatnená na základe porušenia vyšších etických princípov, avšak zodpovednosť by mala byť jednou z primárnych vlastnosti softvérového inžiniera, tak ako je to opísané v bode 2.05.

Zopár otázok na záver

Dokázali Ste si v Martinových potulkách nájsť myšlienky na zamyslenie? Objasňujú Vám niektoré Martinove otázky dôvody zavedenia etického kódexu softvérového inžiniera? Ak nie, ako by Ste si predstavovali Martinove správanie, bez uvedomenia si či je dobré a čo nie? Nie je zdroj tohto rozhodovania len iná forma etického kódexu?

Na spomenutých príhodách sme si ukázali použitie jednotlivých princípov etického kódexu. S jeho obsahom sa plne stotožňujem, aj keď dopad tohto kódexu na praktický život zostáva len na svedomí jednotlivcov, ktorý sa môžu, ale aj nemusia podľa toho správať. Ja osobne sa snažím žiť podľa podobných princípov a hodnôt, a preto verím v dobrý koniec.

Použitá literatúra

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

[7] Wujek, Johnson: How to be a good Engineer, 1/19992 http://onlineethics.org/essasys

[8] McCarthy, J.: Softvérové projekty, Microsoft Press, Praha 1999

[9] ACM Software Engineering Code of Ethics & Professional Practice, IEEE Computer, Oct. 1999, Volume 32, Number 10

Mami, ja som nechcel...!

Roman Filkorn

Dokážeme my, softvéroví inžinieri, v dnešnom komerčnom svete vytvárať kvalitné produkty? Ako nám môžu pomôcť príklady práce ľudí našej society? Koľko sme ochotní obetovať na naplnenie ideálov etického kódexu? Aké máme pravidlá a dôvody takého správania? Na tieto otázky a tretí princíp Etického kódexu softvérového inžiniera [1] reaguje aj táto esej.

Predhovor

Softvéroví inžinieri by mali zaistiť, aby ich produkty a súvisiace modifikácie spĺňali najvyšší možný profesionálny štandard tak, ako je spomenuté v kódexe [1] v princípe 3. Mali by tiež udržovať integritu dát, sledujúc najmä zastaralé a poškodené prípady, pristupovať ku všetkým formám softvérovej údržby s rovnakou profesionalitou ako k novému vývoju a podobne.

Dozrel čas, aby sme sa ako (budúci) softvéroví inžinieri nie len stroho dozvedeli, ale aj vyjadrili svoj názor k pojmu *etika*. Nasmerovaní sme boli k zdroju "najodbornejšiemu". Získali sme *Etický kódex* priamo z dielne ACM, medzinárodnej organizácie združujúcej mnohých odborníkov nami študovanej oblasti. Kódex by nás mal [1] "inštruovať o štandardoch, ktoré od nás societa očakáva, o tom, o čo sa naši rovnocenní partneri snažia a čo očakávajú jeden od druhého".

O čo presne sa teda naši kolegovia snažia? Okrem zabezpečenia svojich životných potrieb, okrem naplnenia svojich životných cieľov, okrem pocitu kreatívnej, slobodnej a hodnotnej práce, okrem dodržiavania profesionálnych, pracovných a spoločenských štandardov vôkol nich, sa naši kolegovia snažia o vysoko ľudskú a napospol filozofickú hodnotu - o *verejný záujem*.

Verejný záujem je ale značne vágny pojem. Len ťažko je určiť hranice, vymedzujúce oba pojmy- verejný, záujem. Kde sa končí verejný nezáujem, začína verejný záujem a neprekračuje sa verejné odsúdenie? Kde je hranica vlastných záujmov, skupinových záujmov a verejných záujmov v najširšom zmysle pojmu verejnosť?

Ospravedlnenie?

V roku 1863 sa Alfréd Nobel [2] zaujímal o transformáciu vysoko výbušného a nestáleho trinitroglycerínu na priemyselnú trhavinu. Vynálezca trinitroglycerínu, chemik Ascanio Sobrero, sám ťažko zdravotne a duševne zasiahnutý, vyhlásil svoj objav za príliš nebezpečný a nepoužiteľný pre civilné (verejné) záujmy. Alfréd Nobel teda nekonal vo verejnom záujme, ba dokonca bol s ním až v príkrom rozpore, obetujúc všetko (pri jednom z pokusov tragicky zahynul jeho brat) pre osobné naplnenie a budúci zisk. Rodina Nobelovcov počas vojen na vynáleze dynamitu obrovsky zbohatla.

Konal pán Nobel eticky? Neporušil jeho *produkt*, dynamit, hneď niekoľko bodov etiky inžiniera (3.01, 3.03, 3.06 či 3.13)? Napriek tomu všetkému bol v časoch mieru naplnený aj verejný záujem: použitie dynamitu na stavebných prácach uľahčilo a zrýchlilo rozvoj stavebných prác. Aké dôsledky (v pokojnej dobe) ospravedlňujú neetické správanie? Snáď prestížne rozdávanie získaného majetku…

Ochrana?

Technickú aj netechnickú verejnosť v nedávnej minulosti rozrušila správa o prvom vrahovi- robotovi. Aféra známa ako Robot - Zabiják ("Killer Robot") [3] vznikla po obvinení jedného z programátorov, spolupodieľajúcich sa na vývoji priemyselného robota. Pôvodné obvinenie stavalo na zistení, že chyba v konkrétnej časti programu (nesprávne interpretovanie fyzikálnych rovníc) zapríčinila tragickú smrť operátora. Počas vyšetrovania prípadu vyplávalo na povrch mnoho závažných skutočností, dotýkajúcich sa vývoja produktu s označením Robbie CX-30. Za všetky spomeňme len tie, ktoré evidentne porušujú Etický kódex:

- Odovzdanie neprebehlo v momente dosiahnutia požadovanej kvality, ale vo vopred dohodnutom termíne bez upozornenia klienta, porušujúc tak bod 3.01.
- V záujme zachovania pracovných príležitostí pre zamestnancov boli sfalšované protokoly o kontrole, porušiac tak body 3.10 a 3.11.
- Počas vývoja užívateľského rozhrania bol použitý vodopádový model namiesto prototypovania, nedodržiac tak body 3.05, 3.06 a 3.07.
- Obvinený programátor nelegálne využíval časti kódu inej firmy, priečiac sa tak bodu 3.13.
- Vedúci projektu nebol dostatočne skúsený pre takýto typ projektu (dlhodobo pracoval na projektoch iného typu), body 3.04 a 3.02.

Všetky body sú hrubým a principiálnym porušením noriem, ktoré by mali spomenutí zainteresovaní v rámci svojej profesie dodržiavať.

Na prípade je ale zaujímavejší moment, keď redaktor časopisu spovedá pracovníkov zákazníckej firmy. Súčasťou objednávky a zakúpenia vyvíjaného robota bola totiž aj povinnosť v určenom rozsahu zaškoliť operátorov a toto potvrdiť písomným testom.

Sami pracovníci (budúci operátori) však absolvovaný test označili za "vtip"- naznačiac tak jeho neadekvátnosť. Pýtate sa, čo by malo byť na takomto postupe zaujímavé? Ak by sa operátori postavili k testu zodpovedne (chrániac tak sami seba), je viac než pravdepodobné, že k usmrteniu by neprišlo. Vychádzajúc z tohto príkladu, môžme povedať, že verejný záujem je záujem ochraňovať (nezodpovednú) verejnosť pred následkami vlastných rozhodnutí?

Adekvátnosť?

Krátko po objavení sa dlho očakávaného produktu s menom Windows 2000 prenikli na verejnosť mimo snahy firmy Microsoft šokujúce informácie [4]. Podľa zdroja by sme za produkt v hodnote päťsto dolárov dostali operačný systém s rádovo šesťdesiattisíc chybami. V záverečných testoch bolo totiž objavených dvadsaťjedentisíc vážnych chýb a ďalších dvadsaťsedemtisíc bolo označených ako *nedokončená* práca alebo *dlhodobo zabudnuté*. Reakcia firmy? Chyby softvéru sú integrálnou súčasťou softvéru, do ich odhalenia sme investovali zatiaľ najviac prostriedkov v histórii firmy…

Konal svetoznámy softvérový gigant eticky? Uvedomme si niekoľko povážlivých skutočností:

- Bol ich plán *zmysluplný* (bod 3.01), keď pred konečným termínom vydania bol tento niekoľko krát posúvaný?
- Bola kvalita ich výrobku *prístupná stanovisku užívateľa a verejnosti* (bod 3.01), keď iniciatíva zverejnenia údajov vyšla z nezávislého (novinárskeho) zdroja?
- Boli ich ciele *náležite stanovené* (3.02) a použili *primerané profesionálne štandardy* (bod 3.06), keď ich výsledkom bolo zarážajúce množstvo chýb? Ak áno, nevypovedá dosiahnutá chybovosť o samotných *profesionálnych šandardoch*?
- Ako sa mohlo stať, že bolo objavených *len* dvadsaťjedentisíc vážnych chýb? Bolo, alebo nebolo zabezpečené *adekvátne testovanie* (3.10)?
- Ešte prekvapivejšie pôsobí označenie dvadsaťsedemtisíc chýb za dlhodobo zabudnuté. Neevokuje naplnenie bodov 3.11 a 3.14 iné dôsledky?

Dotknutá firma na zvýšenie kvality investovala päťsto človeko-rokov a stošesťdesiat miliónov dolárov, zainteresovala sedemstopäťdesiattisíc testovacích osôb. Ako teda zabezpečiť *adekvátnu* kontrolu (body 3.10, 3.11) v intenciách spomenutých investícií? Avšak, ospravedlňuje rozsah investovaných prostriedkov neetickosť predaja? Je vo verejnom záujme nekúpiť si taký produkt, ohroziť existenciu firmy a ňou poskytovanú údržbu starších produktov a de facto ohroziť verejný záujem? Alebo raz bude pompézne odovzdávaná cena Billa Gatesa?

Návod?

Čo ak možný zdroj neetického správania nie je ukrytý v ľuďoch, ale v pravidlách, ktoré majú splniť? Je vôbec možné vytvoriť softvérový produkt vysokej kvality? Je možné, aby sa nejaký program nikdy nezrútil, aby v štyristodvadsaťtisíc riadkoch kódu boli v posledných troch verziách *len* tri chyby, aby sa v posledných jedenástich verziách objavilo sedemnásť chýb (keď *primerané profesionálne štandardy* dosahujú päťtisíc chýb)?

Áno, je to možné [5]. Softvérový tím, zložený z dvestošesťdesiat ľudí, pracuje v rámci spoločnosti Lockheed Martin Corps na projekte, vytvárajúcom riadiaci softvér pre vesmírne rakety. Každý z nich je dospelý človek vymykajúci sa komerčnému obrazu- voľné, športové oblečenie, pizza, kola a nočný pracovný čas- obrazu bežného programátora, akého verejnosť pozná z filmov. Nikto z nich nie je super programátorská hviezda, projekt nestavia na ich nenahraditeľnosti a schopnosti vytvoriť "perfektnú originálnu" časť programu. Vytvárajú taký produkt, akí sú oni sami: dospelí. Vytvárajú taký produkt, aký sa od nich vyžaduje, aký jednoducho musí byť.

Súhlasia pravidlá ich procesu vývoja softvérového produktu s ideálmi Etického kódexu?

Produkt je len tak dobrý ako je dobrý jeho plán

Pred začiatkom *písania* programu venujú analýze a návrhu približne jednu tretinu celkového času. Vytvoria špecifikáciu, ktorá presne hovorí čo, prečo, s čím a ako treba vytvoriť. O každej zmene spíšu rozsiahlu dokumentáciu.

Ale veď 3.07, 3.08 a 3.11 opisujú prvú fázu ich práce!

Najlepšia tímová spolupráca je zdravá rivalita

Súťaživosť medzi tvorcami programu a overovateľmi programu musí zostať na úrovni vzájomnej úcty a profesionálnej cti.

To predsa vytvára ideálne podmienky pre ciele bodov 3.10, 3.11 a 3.04!

Databáza je programová báza

O každej chybe, o každej zmene, o každom návrhu vytvoria *adekvátnu* dokumentáciu. Vždy vedia, čo už bolo opravené, ako, prečo, kedy a s akým výsledkom.

Neopravte len chybu, nájdite aj jej zdroj

Objavenie chyby je určite prínosom pre kvalitu. Ale oveľa viac môže priniesť odhalenie podobných chýb na základe analýzy zdroja už nájdenej.

Našla sa *primeraná metóda* bodu 3.05 a ďalšie aplikácie bodov 3.11, 3.10 a 3.14?

Takže predsa… Podarilo sa nájsť príklad pre úspešný produkt, vznikajúci vysoko profesionálnou (etickou) cestou. Všetko sa zdá zachránené. Všetko bude fungovať. Radosť nám môže pokaziť až reálnejší pohľad na špecifické črty tohto jedinečného produktu:

- *Existuje* úplná špecifikácia požiadaviek, program spracúva vstupy zo senzorov a jeho správanie je prísne určené fyzikálnymi zákonmi a nie je riadené cez žiadne užívateľské rozhranie,
- *Náklady* nie sú problém, keďže projekt je financovaný NASA a môže si dovoliť byť najdrahší v ukazovateli ceny na riadok programu,
- *Konkurencia* prakticky nie je, a teda ani termíny odovzdania nie sú limitným ohraničením,
- Je naozaj riadiaci sofvér rakety vyvíjaný v súlade s etickými normami? Zaručuje *akceptovateľné náklady*?

Myslím, že produkt s takýmito parametrami nepochybne spĺňa všetky profesionálne a etické ciele na najvyššej možnej úrovni a poskytuje nám vzor, ku ktorému by sme sa mali približovať. Jeho prostredie je ale značne iné, než každodenná realita väčšiny vyvíjaných produktov a neposkytuje nám prakticky použiteľný návod. Ako vidíme, porušovanie etických princípov nemožno hľadať v princípoch softvérového inžinierstva, ani v charakteroch samotných softvérových inžinierov, ale v okolitom prostredí, v spoločnosti, v ktorej žijú a chcú prežiť.

Dôvody?

Aké by sme mali mať dôvody pre dodržiavanie Etického kódexu pod tlakom spoločnosti? Primárnym dôvodom nemôže byť kvalita produktu ani produkt samotný, pretože všetky požadované vlastnosti sú určené vonkajšími (spoločenskými) podmienkami. Zdôvodnenie tak isto nemožno nájsť v nutnosti súhlasiť s určenými pravidlami len preto, že ich vyžaduje spoločnosť (societa, ACM a pod.), keďže morálne pozadie vnútenia tak vysoko humánnych a abstraktných pravidiel je značne spochybniteľné (dosiahne sa zvnútornenie sa človeka s danou normou, obohatenie svedomia a tým jeho dobrovoľné a vedomé konanie na základe vnútených pravidiel?). Má teda niekto rozumné dôvody?

Ja svoje mám. A sú tak čisto moje- ako inak, egoistické.

Svedomie!

Verejný záujem sa dá obchádzať len dovtedy, kým si človek neuvedomí, že on sám môže byť (a aj je) súčasťou verejnosti a cieľom naplnenia daných záujmov. Produkcia nekvalitných (alebo- nie vysoko kvalitných) produktov skončí vtedy, keď my sami doplatíme na neprofesionalizmus iných. Keď po trestuhodne ľahkom teste doplatíme na to, že sme nepožadovali test kvalitný. Keď po kúpe päťsto dolárového produktu zistíme, že je v ňom šesťdesiattritisíc chýb (ako by vyzeralo auto s (*len*) tisíc chybami?). Zabezpečenie svojej ochrany veľmi rád prenechám na profesionalizmus niekoho iného.

Návod, ako sa eticky rozhodovať, veľmi zrozumiteľne zhrnuli Grillo a Linderman [6] do nasledujúcich piatich testov:

1. *Mama,* povedali by ste svojej mame, že ste to urobili?
2. *TV,* povedali by ste divákom televízie, že ste to urobili?
3. *Zlý pocit,* máte zlý pocit z toho, že ste to urobili?
4. *V koži iného,* páčilo by sa vám, keby vám to isté spravil niekto iný?
5. *Trh,* bolo by to na trhu dobrým obchodných krokom?

Ak každé naše rozhodnutie nebude v rozpore s vlastným svedomím (resp. jedným z testov), je len otázkou času, kedy začneme vytvárať *kvalitné produkty*. Ak budeme dodržiavať Etický kódex (ktorého body sú len dôsledkami našich doterajších skúseností s kvalitnou tvorbou) a nebudeme jednotlivými ustanoveniami príliš zväzovaní, nikdy nebudeme musieť povedať: Mami, ja som nechcel…!

Použitá literatúra

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

[11] Alfred Nobel - His Life and Work, The Nobel Foundation, http://www.nobel.se/alfred/biography.html, 2. november 1998

[12] Richard G. Epstein: The 'Killer Robot' Scenario, Online Ethics Center, http://onlineethics.org/cases/robot/robot.html, 1998

[13] Mary Jo Foley: Can Microsoft squash 63,000 bugs in Win2K?, ZDNet PCWeek Online, http://www.zdnet.com/pcweek/stories/news/0,4153,2436920,00.html, 11. február 2000

[14] Charles Fishman: They write the right stuff, http://www.fastcompany.com/online/06/nasa.html, december 1996

[15] John Grillo, James Linderman: Ethical Decision Making & Information Technology. An introduction with cases. ISBN 0070340900, Irwin/McGraw-Hill, December 1995, podľa [3]

Hlbší význam

Marcel Bočkay

Témou eseje je štvrtý bod etického kódexu ACM [1] – Úsudok. Zamyslenie nad hlbším neprvoplánovým významom posudzovania.

Pred istým časom som sa zhováral s pracovníkom jednej malej firmy dodávajúcej hardvér a softvér. Hovorili sme o zákazkách a o tom ako ide obchod. Chválil sa mi, že občas sa im vyskytne výborne platená zákazka od akejsi väčšej spoločnosti. Naznačoval, že daná spoločnosť si môže dovoliť platiť, pretože má kontakty – možno aj politické – a pravdepodobne je zapletená aj v neserióznych obchodných praktikách. Tváril som sa prekvapene. Keď to uvidel, zasmial sa a povedal niečo v tom zmysle, že by sa spojil aj s čertom, len keď dobre platia. Ja som zareagoval tak, ako to obvykle robím keď som v rozpakoch a neviem čo povedať. Len som sa pousmial a tváril som sa, že chápem.

Kódex

Posudzovanie či posúdenie (anglicky judgment) v zmysle v akom ho opisuje bod číslo 4. Etického kódexu ACM [1] nemožno chápať len ako povinnosť rozpoznať a posúdiť čo je správne či etické a čo nie. Skôr ide aj o schopnosť správne sa rozhodnúť a podľa tohto rozhodnutia aj konať.

Mohol by som hneď teraz posúdiť prípad, ktorý som uviedol v úvode. Mohol by som naznačiť v čom konkrétne je takéto jednanie v rozpore s etickým kódexom ACM, respektíve aj so všeobecným ľudským etickým kódexom. Tak ako je to riešené napríklad na stránkach „*Onlineethics*“ (Etika online) [2], kde sa uvádza množstvo prípadov z praxe kde došlo k prekročeniu etického kódexu. K jednotlivým prípadom je potom uvedená diskusia a sú naznačené body, v ktorých bolo jednanie neetické t.j. v rozpore s etickým kódexom. Netvrdím, že takéto riešenie nie je správne. Vytvorenie, alebo lepšie povedané zozbieranie príkladov z praxe a ich nasledovné zovšeobecnenie do akýchsi vzorov môže byť veľmi užitočné. Koniec koncov prístup riešenia problémov pomocou vzorov je v informatike v poslednom období populárny napríklad v oblasti návrhu. Pomocou takzvaných návrhových vzorov (anglicky design patterns), ktoré sú zovšeobecnenými prípadmi z praxe sa riešia problémy z podobných doménových oblastí. Tak prečo neskúsiť použiť princíp vzorov aj v inej oblasti? Ide však aj o to, že vzory etického chovania sami o sebe etické chovanie nezaručia.

Človek

Som príliš mladý, aby som mohol hodnotiť situáciu ktorá vládla v našej krajine v nie tak vzdialenej minulosti – za socializmu. Ale aj z rozprávania starších viem, že ak sa v tých časoch vedelo o nejakej nespravodlivosti, často sa o nej vedelo medzi ľuďmi len tajne a potichu. Niekedy aj s istou dávkou strachu. Všeobecne si však mnohí mysleli, že ak by nevládla prísna cenzúra a ak by sa verejne postavila pravda zoči-voči lži, určite by v tejto konfrontácií pravda s jasnou prevahou zvíťazila nad lžou.

V súčasnosti, keď je cenzúra minulosťou a pravda môže byť otvorene konfrontovaná so lžou však vidím, že tento boj vôbec nie je tak jednoznačný, ako by sme si predstavovali. Na obrazovke a v médiách vystupujú ľudia, ktorí si otvorene protirečia a keďže platí, že dva protiklady nemôžu platiť oba súčasne, je jasné, že aspoň niektorí určite otvorene klamú. Je však zvláštne, že aj napriek otvoreným sporom majú všetky strany vždy svojich prívržencov, ktorí sú presvedčení o ich pravde.

Iste je správne, že máme k dispozícií pluralitu názorov. A rozumný jedinec si môže a má utvoriť vlastný uvážený názor. To na čo chcem poukázať je fakt, že aj napriek otvorenosti konfrontácie môže byť rozdiel medzi správnym a nesprávnym len tenkou čiarou, ktorá je často zahmlená a treba ju stále hľadať. Nemožno ju vyznačiť nejakými presnými pravidlami, podľa ktorých by bola vždy a na prvý pohľad jednoznačne rozpoznateľná. Nemožno teda zostrojiť ani dostatočné vzory, ani dostatočný algoritmus, ktorý by za každých okolností rozhodol čo je etické a čo nie. Je to tak preto, že opisovaná realita je príliš zložitá. Nemyslím si však, že preto nemá cenu zaoberať sa posudzovaním dobra a zla. Človek je totiž tiež mnoho a mnohokrát zložitejší, ako len nejaký predpis či algoritmus, a aj preto má možnosť posudzovať správnosť svojho konania.

Definície

Aby sa niektoré kľúčové pojmy správne chápali, treba ich v predpisoch často všeobecne definovať. Takéto definície možno nájsť aj na už spomínaných stránkach „*Onlineethics*“ [2].

Úplatok je definovaný ako niečo, čo je darované alebo poskytnuté osobe alebo organizácií v zodpovednom postavení, aby sa táto chovala v rozpore s danou zodpovednosťou. Úplatky sú zvyčajne vyplácané z dôvodu získania niečoho, na čo daný žiadateľ nemá právo, ako napríklad zvláštne výhody pri získavaní kontraktu. Je však dôležité rozlišovať úplatok od vydierania. Pri vydieraní je platiaca strana prinútená zaplatiť v dôsledku hrozby za niečo, na čo má právo aj tak.

Ďalším z kľúčových pojmov je *konflikt záujmov*. Osoba sa nachádza v konflikte záujmov v prípade, že je na pozícií, ktorá vyžaduje jej posúdenie v záujme ostatných (ľudí, inštitúcií, atď.) a zároveň má záujem alebo záväzok, ktorý môže zasahovať jej posudzovanie. Od osoby sa morálne vyžaduje buď odstrániť konflikt, alebo otvorene ho zverejniť. Slovníkové definície často uvádzajú tento pojem iba v prípade konfliktu

medzi súkromným záujmom osoby a záujmom organizácie, ktorý môže byť touto osobou ovplyvnený. Existujú však aj konflikty záujmov, kde nevystupujú priamo záujmy konkrétnej súkromnej osoby. Napríklad ak firma poskytuje svoje služby dvom spoločnostiam, ktoré si konkurujú na spoločnom trhu. Inžinierske firmy by mali byť zvlášť opatrné, aby nedošlo k prezradeniu informácií o jednej spoločnosti inej.

Na stránkach ICAC (Independent Commission Against Corruption) [5], ktoré poskytujú rôzne publikácie týkajúce sa korupcie a opisujú aj možnosti, ako postupovať v prípade konfliktu záujmov, možno nájsť podobnú definíciu. *Konflikt záujmov* vzniká, ak osoba, ako zamestnanec alebo predstaviteľ verejného sektoru, je pri vykonávaní svojho povolania ovplyvnená alebo sa zdá byť ovplyvnenou osobnými záujmami.

Ani takéto definície však nezaručia, že v praxi nebudú pojmy prekrúcané a interpretované tak, aby sa dali obísť. Ľudia často radi tvoria a využívajú aj rôzne skryté formy ako úplatkov, tak aj konfliktu záujmov. Zúčastnené strany sa potom sami aj vzájomne presvedčujú, že vlastne ide o niečo úplne iné. Samozrejme etické.

Ako istú formu úplatkov možno chápať napríklad aj rôzne semináre, kde sa prezentujú služby firmy a pri tej príležitosti sa poskytujú účastníkom seminára rôzne dary. To samozrejme závisí od typu a hodnoty darov. Čo sa týka konfliktu záujmov je napríklad zložitou otázkou prípad, keď sa inžinier zamestná v súkromnom sektore v oblasti, v ktorej bol predtým zamestnaný vo verejnej sfére, napríklad v nejakej vládnej organizácií. Takýto pracovník môže mať znalosti zo svojho predošlého zamestnania, o ktoré majú záujem súkromní klienti, ale tieto informácie nie sú verejne prístupné.

A tu sa dostávam k tomu, prečo som povedal, že vzory sami o sebe nestačia. To čo je a čo nie je etické či správne, nemožno tak jednoducho a pritom komplexne a formálne opísať, ako v iných technických oblastiach. Vynaliezavosť človeka tvoriaceho predpisy mu umožňuje v prípade potreby ich aj úspešne obchádzať. Je potrebné nielen mať k dispozícií etické kódexy, ale aj pochopiť ich hlbší zmysel.

Možno by sme si preto mali položiť aj otázky, či sme naozaj schopní posúdiť, čo etické je a čo nie? Čo vlastne etika je a aký má zmysel a význam chovať sa eticky? Nebolo by lepšie vôbec sa etikou nezaoberať? Tieto otázky sa môžu zdať príliš všeobecné, napriek tomu sú rozhodujúce pre následné posudzovanie v konkrétnych prípadoch. Keď totiž nevyriešime tieto otázky aspoň sčasti, stanú sa akokoľvek prešpekulované a komplikované predpisy a etické kódexy len bezcenným obalom, ktorý sa v prípade potreby použije, alebo naopak odhodí.

Etika

Vladimír Neff vo svojej knihe „Filosofický slovník pro samouky, neboli Antigorgias“ [3] definuje etiku ako tretiu najdôležitejšiu filozofickú disciplínu. Etika (grécky éthiké; éthos = zvyk, mrav, chovanie) sa zaoberá praktickými otázkami aký má jedinec zaujať postoj ku skutočnosti. Ako sa má chovať a prečo sa má tak chovať. Etika je filozofia praktická.

Život každého človeka sa dotýka množstva noriem, príkazov a predpisov, ktoré ovplyvňujú jeho jednanie a podľa ktorých bývajú jeho činy hodnotené pozitívne, alebo negatívne. Napríklad trestný zákonník, spoločenský katechizmus, náboženské prikázania, ale aj pracovné a úradné nariadenia. Vžité zvyky, občianske, rodinné a profesionálne povinnosti a tak ďalej od „nezabiješ“ až po „zákaz vstupu“. Niektoré z týchto noriem môžu variovať s časom, hospodárskymi podmienkami a s národmi. Etika však musí byť samostatná (autonómna). Vecou samostatnej etiky nie je, aby stanovila či mám zaplatiť dane alebo nie. Jej jedinou úlohou je nájsť princíp, ktorý by človeku umožnil presne a neomylne rozlíšiť dobré od zlého. Inými slovami, úlohou etiky je nájsť vždy, za každých okolností a pre všetkých platné kritérium mravného a nemravného. Pri hľadaní takéhoto kritéria je dôležité stanoviť, čo považujeme za najvyššiu hodnotu ľudského života. A touto hodnotou je iste dobro a nie zlo.

Zákon ľudskej prirodzenosti

Otázkou však ostáva, aký význam má chovať sa eticky, alebo či etické chovanie je človeku nejakým spôsobom prirodzené alebo nie?

Odpovede načrtáva americký spisovateľ a profesor renesančnej literatúry na univerzite v Cambridge C. S. Lewis (1898-1963) v knihe „K jádru křestanství“ [4]. Poukazuje na chápanie dobra a zla, ktoré je vlastné každému človeku. Toto chápanie nazýva „zákonom ľudskej prirodzenosti“. Tento zákon možno nájsť už v základoch ľudského správania. Každý už počul ľudí hádať sa. Hovoria veci ako: „Ako by sa ti asi páčilo, keby niečo také urobil niekto tebe?“ – „To je moje miesto, ja som prišiel prvý.“ – „Nechaj ho tak, nič ti predsa nerobí.“ – „Prečo by si sa tam ako prvý mal napchať ty?“ – „Požičaj mi peniaze, ja som ti predsa tiež požičal“ – „To nemyslíš vážne, veď si mi to sľúbil.“ Také veci hovoria ľudia denne, jednoduchí rovnako ako učení, deti rovnako ako dospelí.

Na týchto poznámkach je zaujímavé, že ten kto ich vyslovuje, nehovorí nimi len to, že mu chovanie druhého človeka nepôsobí žiadnu radosť. Dovoláva sa určitého štandardu chovania, ktorého znalosť predpokladá aj u druhého. Málokedy na to dostane odpoveď: „S tým tvojím štandardom sa môžeš dať vypchať.“

Skoro vždy sa mu ten druhý snaží dokázať, že jeho konanie v skutočnosti nie je s týmto štandardom v rozpore. Alebo, že má preň nejaké zvlášť mimoriadne ospravedlnenie. Tvrdí, že z nejakého mimoriadneho dôvodu a práve v tomto konkrétnom prípade má si ten, čo obsadil miesto ako prvý, sadnúť inde. Alebo, keď kontrakt uzatváral on, mali sa veci celkom ináč, alebo sa prihodilo niečo, čo ho teraz z povinnosti dodržať sľub vyväzuje. Vlastne to vyzerá, ako keby sa obaja zúčastnení dovolávali akéhosi zákona, či pravidla poctivosti, slušného správania, mravnosti, či akokoľvek inak to chcete volať, na ktorom sa skutočne dohodli. A tak to aj je. V opačnom prípade by spolu mohli zápasiť ako zvieratá, ale nemohli by sa hádať v ľudskom zmysle slova. Hádať sa znamená pokúšať sa druhému človeku ukázať, že sa mýli. To by nemalo zmysel, keby medzi nami a tým druhým neexistovala nejaká dohoda o tom čo je správne a čo

nesprávne. Rovnako by nemalo zmysel povedať, že futbalista urobil faul, keby neexistovala dohoda o tom, podľa akých pravidiel sa futbal hrá.

A práve tento zákon, či pravidlá o správnom a nesprávnom sa volal „zákon prirodzenosti“. Dnes sa však hovorí skôr o „zákonoch prírodných“ a myslí sa napríklad zemská príťažlivosť, dedičnosť alebo chemické zákony. Podľa Lewisa však starí myslitelia hovorili o zákone správneho a nesprávneho a volali ho „zákonom prirodzenosti“ mieniac tým „zákon ľudskej prirodzenosti“. Jeho jadrom bola myšlienka, že rovnako ako sa teleso riadi zákonom zemskej príťažlivosti alebo ako sa organizmy riadia zákonmi biologickými, má svoj zákon aj tvor menom človek – avšak s tým podstatným rozdielom, že teleso zákonu gravitácie podlieha, či chce alebo nechce, kým človek môže zákon ľudskej prirodzenosti dodržiavať, či porušovať podľa vlastnej voľby.

Zákonu zemskej príťažlivosti, ani zákonom biologickým sa človek nemôže vzoprieť o nič viac ako zviera, či neživý kameň. Človek aj kameň budú vo voľnom priestore bez opory padať rovnako. Zákon, ktorý môže človek slobodne neposlúchnuť, platí len pre prirodzenosť ľudskú.

Sú ľudia, ktorí budú namietať, že myšlienka zákona prirodzenosti alebo slušného chovania, ako čohosi známeho všetkým ľudom je neopodstatnená, lebo v rôznych civilizáciách a rôznych dobách platili rôzne mravy. To však nie je pravda. To, že sa mravné zásady navzájom líšia, nikdy neznamená naprostú odlišnosť. Ak si dá niekto tú prácu a bude porovnávať mravné poučenia napríklad starých Egypťanov, Babylončanov, Hindov, Číňanov, Grékov a Rimanov, bude prekvapený, ako sa podobajú jedny druhým a tiež našim. Ľudia sa môžu líšiť v názoroch na to, ku komu by sa mal človek správať nesebecky. Či len ku svojej rodine, či ku svojim krajanom, alebo ku každému. Vždy sa však dohodnú, že na prvé miesto by človek nemal postaviť seba samého. Sebectvo zdrojom obdivu nikdy nebolo. Ľudia sa môžu nezhodovať v tom, či má muž mať jednu ženu alebo štyri. Vždy sa však zhodli na tom, že nemôže mať ktorúkoľvek ženu, ktorú si zaželá.

Pozoruhodné je však toto: môžete stretnúť človeka, ktorý dané slovo síce sám nedodrží, ale keď ho nedodržíte vy, bude si sťažovať, že to nie je fér. Podobne by mohol byť národ, ktorý prehlasuje, že na zmluvách nezáleží, ale v nasledujúcom okamžiku bude tvrdiť, že tá zmluva, ktorú hodlá porušiť nie je spravodlivá. Ak na zmluvách nezáleží, ak neexistuje správne a nesprávne, ak neexistuje zákon prirodzenosti, aký je potom rozdiel medzi zmluvou spravodlivou a nespravodlivou? Neprezrádza tým skôr hovorca práve, že napriek všetkému v skutočnosti zákon prirodzenosti pozná ako každý iný.

Ďalším zaujímavým výsledkom, ku ktorému Lewis dospieva je, že čovek je v skutočnosti pod tak silným tlakom tohoto zákona, že jeho porušovanie nedokáže uniesť a snaží sa zodpovednosť preniesť na niekoho iného.

Zamyslenie nad zákonom ľudskej prirodzenosti nám prináša odpoveď na otázku, či sme schopní posúdiť a rozlíšiť čo je správne a čo nie. A navyše sa ukazuje, že sme nielen schopní rozlíšiť etické od neetického, ale navyše v hĺbke svojho vnútra cítime,

že sa máme rozhodnúť pre etické konanie. Toto rozhodnutie je však pre každého dokonale slobodné.

Dôvody etického chovania

Každý sa už určite stretol s názorom, že chovať sa poctivo či eticky sa v dnešnej dobe voľného trhu a komercie nevypláca. Keď chce niekto preniknúť, alebo niečo získať, musí sa chovať dravo a určite neeticky. Nuž, dôvody prečo sa chovať eticky by som rozdelil do dvoch kategórií. Jedny by som nazval *prvoplánové* a druhé *dlhodobé*, alebo *skryté*. Prvoplánové dôvody si uvedomuje pravdepodobne každý. Aj spoločnosť s liberálnym prístupom k trhu stanovuje zákonné obmedzenia a postihy za ich porušovanie. Úplatkárstvo je okrem etického priestupku aj trestný čin. Takisto aj konflikt záujmov možno riešiť právnou cestou.

Asi pred pol rokom (07.06.1999) uviedla agentúra Reuters [6] správu, že Švajčiarsko má poslať Argentíne 4,5 milióna amerických dolárov uložených vo Švajčiarskych bankách. Tieto peniaze mali byť časťou $21 miliónového úplatku, ktorý mala zaplatiť firma IBM viacerým členom Argentínskej vlády a predstaviteľom Argentínskej Národnej Banky v roku 1993, aby tak získala $250 miliónový kontrakt na inštaláciu informačného systému pre najväčšiu Argentínsku banku Argentina's Banco de la Nacion. Podozriví na americkej strane odmietli svedčiť v Argentíne a IBM jednoznačne popiera akékoľvek podvody v súvislosti so spomínaným kontraktom. Toto je len jeden z mnohých prípadov, keď podozrenie z korupcie prerastie aj do oficiálneho vyšetrovania a každému je jasné, že pre všetky strany zapletené do takejto aféry je to nevýhodné nielen po etickej, ale aj finančnej stránke. Anglický pojem „goodwill“ s rozvíjajúcim sa zahraničným obchodom preniká aj na Slovensko. „Goodwill„ znamená niečo ako „dobré meno“ a predstavuje finančne oceniteľnú hodnotu. V zahraničí existujú renomované agentúry, ktorých náplňou práce je zhromažďovať údaje o firmách a tieto komerčne poskytovať záujemcom zo zahraničia, ktorí majú záujem s tou ktorou firmou uzatvoriť kontrakt a chcú si overiť jej serióznosť. Snáď už ani netreba dodávať, že finančné aféry a škandály k tejto serióznosti nijako neprispievajú. A tak, ako môžu kaziť neseriózne praktiky meno firme, môžu nepriaznivo vplývať aj na povesť celej profesie.

Som však presvedčený, že existujú aj iné hlbšie dôvody prečo byť etický. Tieto iné dôvody niekedy nie sú také viditeľné. Často najmä preto, že sa nemusia ukázať hneď, čo však neznamená, že neexistujú. Človek s chorým žlčníkom by si možno rád dal na obed pečenú hus, lenže aj keď by mu to prinieslo dočasné uspokojenie a teda relatívny zisk, neurobí to, pretože by za to neskôr zaplatil oveľa viac. Podobne je to aj s takzvanými skrytými dôvodmi prečo byť etickým, len s tým rozdielom, že nemusíme vedieť kde, kedy a ako sa neetický prístup ukáže ako nevýhodný. Ľudia už dávno vypozorovali aj takéto zákonitosti, o čom svedčia porekadlá, ako napríklad: „Božie mlyny melú pomaly, ale isto“.

Treba však povedať, že pre tento hlbší pohľad na etiku treba mať možno občas aj istú dávku idealizmu (nie však naivity). Pretože vieme, že aj v našej spoločnosti boli, sú a

budú ľudia, ktorí sa zachovali neeticky a nielenže neboli potrestaní, ale darí sa im a vo svojej „podnikateľskej“ činnosti pokračujú. A niekedy sa zdá, že niet nikoho a ničoho, čo by ich zastavilo a potrestalo, a uviedlo tak veci na správnu cestu. Neverím však, že človek je len stroj na peniaze a je mu vnútorne jedno, ako sa k nim dostane, pretože či si to chceme pripustiť alebo nie, náš život sa skladá ako s telesnej, tak aj so psychickej či duševnej časti. K téme idealizmu uvádzam ešte citát Alberta Schweitzera (1875-1965) [7] rakúskeho hudobného skladateľa, spisovateľa a lekára, ktorý je známy aj svojim veľkým úsilím, ktoré venoval chorým v Afrike.

„V mladosti som počúval rozhovory dospelých, z ktorých mi jeden smútkom zovrel srdce. Spomínali na idealizmus a na to, ako sa v mladosti dokázali nadchnúť, ako na niečo cenného, čo si mali udržať. Súčasne to ale považovali za akýsi prírodný zákon, že si to nemožno uchovať.

Vtedy som dostal strach, že sa raz budem musieť tiež tak žalostne pozerať späť na svoj život. Predsavzal som si, že sa nepoddám tomuto tragickému vývoju k *zrelosti*. Čo som si takmer v chlapčenskom vzdore zaumienil, snažil som sa uskutočniť.

Dospelí radi nachádzajú záľubu v smutnej úlohe pripravovať mladých ľudí na to, že raz budú mnohé z toho, čo teraz povznáša ich srdce a myseľ, považovať za ilúziu. Ale hlbšia skúsenosť hovorí k neskúseným inak. Zaprisaháva mládež, aby si pevne udržala po celý život myšlienky, pre ktoré je nadšená. V idealizme mladosti nachádza človek pravdu. V ňom má bohatstvo, ktoré nemá za nič vymeniť.

Musíme počítať s tým, že nás život bude chcieť pripraviť o vieru v dobro a pravdu i o nadšenie pre ne. Ale nesmieme svoju vieru a nadšenie zahadzovať. Naše ideály bývajú v strete so skutočnosťou potlačené, ale to neznamená, že majú kapitulovať.“

Záver

Pri uvedomení si hlbšieho významu zákonov, etických kódexov a zodpovednosti pri posudzovaní a rozhodovaní sa stanú zákony tým, čim majú byť. Teda nie zákonmi pre zákony, zákonmi ktoré sú na to aby sa porušovali, ale pre ľudí, ako pomôcka pre dosiahnutie cieľa, pre ktorý boli vytvorené. Proces etablovania profesie sofvérového inžiniera sa vo svete aj u nás už začal. Uvážený prístup k posudzovaniu a rozhodovaniu v rámci tejto profesie tento proces určite urýchli.

Použitá literatúra

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

[17] Onlineethics, URL: http://www.onlineethics.com, (23/3/2000).

[18] Vladimír Neff, Filosofický slovník pro samouky neboli Antigorgias, Mladá fronta / Kdo je kdo, Praha 1993, ISBN 80-204-0383-3.

[19] C. S. Lewis, K jádru kresťanství, Návrat, Praha 1993, ISBN 80-85495-19-8.

[20] ICAC - Independent Commission Against Corruption, GPO Box 500, Sydney, New South Wales, Australia, 2001, URL: http://www.icac.nsw.gov.au, (23/3/2000).

[21] Reuters (06/07/99, 6:49 a.m. ET), Možno nájsť aj na URL: http://www.techweb.com, (23/3/2000).

[22] Albert Schweitzer, Zastánce kritického myšlení a úcty k životu, Vyšehrad, Praha, 1989, ISBN 80-7021-010-9.

Súdnosť v softvérovom inžinierstve, alebo „Linux versus Windows“

Peter Burger

Táto esej sa zaoberá časťou Súdnosť (Judgement) *Etického kódexu softvérového inžiniera [1]. Ukazuje ako sú dve pravidlá z časti* Súdnosť *často porušované pri posudzovaní operačných systémov Linux a Windows.*

Linux versus Windows

V súčasnosti má prevahu na trhu s operačnými systémami pre komerčné využitie prím firma Microsoft s jej produktom Windows 9X/NT. V dobách, keď ešte bol Microsoft „v plienkach“ existoval operačný systém UNIX, ako asi najrozumnejší a najrozšírenejší operačný systém. Lenže v tých dobách sa počítače používali najmä vo výskume, čiže na univerzitách a laboratóriách firiem. Potom prišiel, dnes všetkým známy, pán Bill Gates, ktorý mal, a dodnes zrejme má, podnikateľskú víziu. Tá vízia bola „Počítač na každom pracovnom stole“ – pochopiteľne s jeho softvérom. Na tejto vízii pracoval a myslím, že sa mu to z veľkej časti podarilo. Prečo postupoval, tak ako postupoval? To vie asi len on sám, ale asi sa nebudem veľmi mýliť v tom, keď si myslím toto: V tých dobách keď v Microsofte vytvorili MS-DOS, ktorí sa s UNIX-om nedá porovnávať, nebola taká technika, aby každý mohol mať na stole stroj s UNIX-om. A Microsoft *išiel* za víziou „Počítač na každom pracovnom stole“. Takže aj softvér musel byť prispôsobený tak, aby sa dal prevádzkovať na počítačoch, ktoré mohli byť na každom stole. Ďalej v Microsofte vedeli, že ak sa má výpočtová technika rozšíriť medzi normálnych ľudí, teda nie ľudí z akademickej obce alebo spomedzi odborníkov, musí byť softvér prispôsobený tak, aby sa ľahko ovládal, bolo ho treba čo najmenej udržiavať a nemusel byť až taký spoľahlivý - predsa len, keď niekomu spadnú Windows, tak ich spustí znovu a pracuje ďalej, čo je samozrejme nemysliteľné v prípade UNIX-ového servera, ktorý obsluhuje 100 používateľov.

Toto viedlo k tomu, že v Microsofte vyvíjali softvér pre ľudí tak, aby slúžil hlavne ľuďom. UNIX išiel svojou cestou, cestou operačného systému, ktorý je určený práve pre servery a aplikácie, ktoré vyžadujú spoľahlivosť. V tejto oblasti netreba vymoženosti typu okienka, myš, vizuálne ovládanie a podobne.

Technika sa vyvíjala a časom sa stalo, že UNIX-ové systémy už mohli byť prevádzkované na počítačoch, ktoré boli na „každom pracovnom stole“. Ale kto by na ne inštaloval UNIX-ový operačný systém, ktorý nemal prakticky žiadne programové vybavenie potrebné v kancelárii? Písanie korešpondencie, tvorba kalkulácií, tvorba prezentácií... to bola doména Microsoftu, ktorý na aplikáciách tohto typu pracoval roky. Taktiež UNIX-u chýbalo používateľsky príjemné rozhranie, veď kto by pracoval v prostredí s čiernou obrazovkou a zobáčikom v ľavom dolnom rohu? Oveľa lepšie bolo súbory premiestňovať myšou, alebo ťuknutím na F5 a stlačením ENTER-u.

Jedného dňa však nastala zmena. Pán Linus Torvalds napísal UNIX-oidný operačný systém, ktorý nazval Linux. A práve tu sa začínajú spory o tom, čo je lepšie, či Linux, alebo Windows (pod Windows rozumiem Windows NT, pretože Windows 9X sa nedajú s Linuxom porovnávať). Na začiatku to bol tzv. „komandpromptový“ operačný systém. Rozšíril sa medzi nadšencami z dvoch dôvodov: bol prakticky *zadarmo* a dal sa prevádzkovať na strojoch, ktoré boli v tom čase *„na každom pracovnom stole“*. Otázka je PREČO bol zadarmo? Pretože ho vytvorili dobrovoľníci/nadšenci. Neskôr vznikli rôzne implementácie grafického používateľského prostredia (GNOME, KDE, FVM95,...). Je zaujímavé, že prebrali väčšinu know-how od večne kritizovaného Microsoftu, ktorý vyvíjal veľmi veľké úsilie nato, aby vyvinul príjemné používateľské rozhranie (pre Linuxákov: toto nie je kritika, to je konštatovanie).

V poslednom čase sa Linuxu „chytili“ aj veľké softvérové firmy, ktoré vyvinuli vlastné distribúcie Linuxu (napr. Corel Corporation). Videl som ich a musím povedať, že podľa môjho názoru ich snaženie je na najlepšej ceste spraviť Linux použiteľný pre normálneho používateľa.

Výsledok: Linux začína konkurovať Windows NT a počítačový svet sa rozdelil na dve polovice: LinuxMan-i a WindowsMan-i.

Čo s týmto všetkým má spoločná *Súdnosť* v softvérovom inžinierstve?

Spomínané skupiny sa navzájom očierňujú a predháňajú v argumentoch prečo je ten ich systém lepší než konkurenčný. Niekedy mi to celé pripomína hádanie detí na ihrisku, že kto má lepšie a krajšie formičky. Pritom sú to obyčajne vysokoškolsky vzdelaní ľudia, od ktorých ja osobne čakám rozumné uvažovanie. A pritom keď si pozriete niektoré ich argumenty, tak jedna strana ich použije vo svoj prospech a druhá strana ten istý argument vyloží vo svoj prospech.

Microsoft v októbri minulého roku (1999) zverejnil na Internete materiál nazvaný „Linux myths“, čiže mýty o Linuxe. V [2] bol publikovaný článok, kde autor (Martin Devera, ďalej len MD) reaguje na tieto tzv. mýty. Treba pripomenúť, že jeho reakcie boli silno prolinuxové.

Príklady a prehrešky:

Príklad č. 1

MS:

„Linux nie je stavaný na SMP (symetrický multiprocesor), ani na GUI, nemá asychrónne operácie, jeho technológia je stará 30 rokov ...“

Prehrešok:

Informácia o dĺžke života operačného systému je irelevantná vzhľadom na jeho funkčnosť a výkon. Z veku architektúry operačného systému sa predsa nedá posúdiť či je dobrý alebo nie. Nakoniec, autá chodili už viac než pred 100 rokmi na 4 kolesách a pritom máme aj dnes moderné a výkonné autá. Microsoft manipuluje čitateľa, nedodržiava *objektivitu*, prikázanie č.3.

MD:

„Microsoft o Linuxe hovorí, že je zastaralý, a pritom do Windows doplňuje funkcie, ktoré sú v UNIX-och už dávno implementované.“

Prehrešok:

Microsoft nepovedal, že jednotlivé funkcie operačného systému sú zastaralé, ale že *technológia* operačného systému je zastaralá. Pán MD tak ako Microsoft nedodržiava *objektivitu*, prikázanie č.3.

Príklad č. 2

MS:

„Linux nie je výkonnejší než Windows NT“.

MD:

„Týmto asi Microsoft myslel testy porovnávajúce výkon WinNT a Linux, ktoré ukazujú, že WinNT sú asi 3-krát rýchlejšie ako Linux. Dôvod tohto rozdielu je chyba v jadre Linux-u, ktorá je opravená od verzie 2.3. Chyba sa týkala SMP. Linuxová podpora SMP ešte nie je doladená. Na jednoprocesorovom systéme by Linux pravdepodobne predčil WinNT.“

Prehrešok:

MD celú svoju úvahu stavia na tom, čo si on sám domyslel, že čo Microsoft myslel – dosť odvážne. Ak to však bola pravda, potom Microsoft vo svojom výroku „zabudol“ spomenúť, že sa jednalo o SMP systém a že vlastne problém nie je v technológii Linux-u, ale v chybe operačného systému. Opäť Microsoft nedodržal *objektívnosť*.

Príklad č. 3

MD: (pokračovanie predošlého problému)

„Je tu jeden podstatný rozdiel. Internet Information Server pod WinNT používa multithreadové riešenie, ktoré je výkonnejšie, ale zároveň menej stabilné. Server Apache v Linux-e nepoužíva multithreadové riešenie, práve kvôli stabilite.“

Prehrešok:

Neviem ako vy, ale pokiaľ viem ja (opravte ma ak sa mýlim, e-mail je v hlavičke eseje), tak Linux *nemá* multithreadovú podporu. Takže možno práve preto webserver Apache ju nepoužíva ;–) a nie preto, že je to nestabilnejšie. Komické však?

Príklad č. 4

MS:

„Prieskum ukázal, že 93% profesionálnych vývojárov vyvíja aplikácie pod WinNT, a pre Linux iba 13%.

Prehrešok:

Jasné, že WinNT vývojárov je väčšina, keď viac je rozšírený operačný systém WinNT. To ako keby som povedal, že väčšina ľudí tankuje do svojho auta benzín a preto sú lepšie autá na benzín ako na plyn. Ale však máme aj autá na plyn, ktoré napríklad menej znečisťujú životné prostredie. Opäť Microsoft nedodržal *objektivitu*.

MD:

„Skôr povedzme, že pre Windows je viac aplikácií jednoducho preto, že sú rozšírenejšie – a to preto, lebo pre laika sú jednoduchšie na inštaláciu.“

Prehrešok:

Logická chyba: viac áut na benzín je preto, lebo sú rozšírenejšie :-). Okrem toho MD sám priznáva, že správa Linux-u je obtiažnejšia a pre laika nepríjemná. Neviem prečo dáva argumenty, ktoré môžu byť obratom použité proti nemu samému.

Takýchto príkladov sa dá nájsť oveľa viacej. Sú vlastne nevinné a novinári sa len môžu tešiť, že je o čom písať a webmasteri, že je čo „uploadovať“. A tí čo majú vlastný rozum sa nad nimi iba pousmejú a pokračujú ďalším článkom a nechajú detičky, na už spomenutom ihrisku, hádzať si piesok do tváre. Ako to vidím ja?

Ja to vidím tak, že Linux a Windows NT sa zatiaľ dajú porovnávať iba na úrovni architektúry a mám pocit (hovorím pocit, nemôžem to ničím doložiť), že WinNT sú na tom lepšie. Sú mladšie, čo im dáva lepšiu šancu využívať moderné technológie, ktoré v UNIX-e jednoducho nemôžu byť aplikované, napríklad kvôli spätnej kompatibilite. Prirovnal by som to k vývoju procesorov INTEL X86. Ich rovesníci boli vždy výkonnejší a vedeli adresovať viacej pamäti a to všetko kvôli zastaranej architektúre a spätnej kompatibilite. Zároveň tým, že sú mladšie, majú viacej „múch“, ktoré bude treba vychytať a ktoré UNIX už prekonal. Až Linux dospeje na úroveň WinNT, čo sa týka používateľskej podpory, aplikačnej bázy, pohodlnosti, podporovaného hardvéru,

potom budeme môcť porovnávať, ktorý z nich je lepší a v čom. Nejde predsa porovnávať čiernobiely monitor s farebným.

Rasizmus medzi softvérovými inžiniermi – čo nato ľudské hodnoty?

Raz sme sa s kolegom zo školy bavili na tému Linux versus Windows. Skoro som sa „na zadok posadil“, keď som sa dozvedel, že aj Linux sa dá naučiť a tí hlúpejší čo to nevedia, nech používajú Windows. Už chýba len malý krôčik k tomu, aby sme začali ukazovať na ľudí prstom, že on je ten čo používa Windows, to je ten čo je hlúpejší.

Ja osobne používam radšej WinNT, pretože mi dávajú to čo potrebujem. Sú mojím nástrojom, či už pri vývoji, alebo pri korešpondencii. Existuje veľké množstvo zrozumiteľnej dokumentácie k vývojovým nástrojom pre WinNT. Jednoducho, to čo potrebujem väčšinou aj mám. Toto v mojom prípade *neplatí* pre Linux. Som snáď preto hlúpejší ako tí čo používajú Linux? Mimochodom, nepoznám jediného „Linuxáka“, ktorí má iba Linux, lebo mu stačí. Každý z nich má aj Windows, lebo inak by nemohol komunikovať so svetom. Teraz každý z Vás Linuxákov fučí, lebo „veď toto je práve náš argument“. Omyl! Urobte si softvér, ktorý vie komunikovať zo svetom. Ak si kúpite auto na plyn, tak nebudete môcť odísť s tým autom ďalej od plynovej čerpacej stanice ako je „dolet auta“. Jednoducho si musíte nainštalovať „importné filtre na benzín“, aby ste mohli žiť s ostatnými, ktorí chodia na benzín, lebo ináč budete to auto tlačiť. Niektoré firmy to už pochopili a snažia sa z celej sily o to, aby softvér bol kompatibilný na rôznych platformách. Je to otázka postoja ľudí (vás softvéristov – či už linuxákov alebo windowsákov) k spoločnosti/ostatným kolegom a to je otázka *posudzovania* reality.

Stalo sa mi na jednom cvičení v piatom roku štúdia, že keď som sa spýtal cvičiaceho, že ako sa robí to a to v UNIX-e, tak som sa dozvedel, ohromne „múdru“ odpoveď, že to mám vedieť, že však som mal ten a ten predmet v druhom roku štúdia. A tak som sa dozvedel ako som ja hlúpy, pretože si nepamätám, ako sa nejaká vec robí v operačnom systéme, ktorý nepoužívam a naposledy som s ním pracoval pred 3 rokmi. Toto nie je ojedinelý prípad. Mám viacej známych, ktorý pracujú pod UNIX-om, nie profesionálne, ale používateľsky, ktorí mi signalizovali podobný jav. Keď niečo nevedeli a spýtali sa, dostali späť signály, že ako je možné, že to nevedia a že v manuály to predsa všetko je napísané, takže čo vlastne chcú. Toto je typická choroba UNIX-menov.

Ďalším prehreškom voči etike pri posudzovaní je voľba operačného systému pre danú aplikačnú oblasť. Často sa stáva, že pri otázke voľby platformy sa často rozhoduje podľa iného kritéria ako toho, že čo je lepšie pre používateľa, alebo čo potrebuje používateľ. Raz som sa dopočul od jedného „systemáka“ u nás v škole výrok „Keby bolo po mojom, tak je tu všade Linux!“. Tohto správcu asi vôbec nezaujímalo čo potrebujú používatelia, teda my študenti, ale čo by on chcel, aby bolo nainštalované.

Čo na záver?

Mám pocit, že WindowsMan-i trochu preháňajú keď každú „feature“ predstavujú ako vlastnú a novú i keď UNIX ju mal už v prehistorických dobách a LinuxMan-i by si mohli zvyknúť, že zato, že vedia pracovať s Linuxom, ešte neznamená, že sú nadľudia.

Použitá literatúra:

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

[24] Martin, D.: Mýty o Linuxu, nebo mýty o Windows?, Chip 04/00 str. 154, Vogel Publishing, s.r.o.

Etika ako princíp pozitívnych hodnôt

Filip Pucher

Etické princípy v softvérovom inžinierstve a iných disciplínach ľudského počínania si zasluhujú čoraz viacej pozornosti. Táto práca si kladie za cieľ oboznámiť čitateľa s jedným z pohľadov na uplatňovanie etiky v praxi a reagovať tak na Etický kódex softvérového inžiniera [1].

Úvod

S pojmom etika sa nestretávame po prvý raz. V našom predchádzajúcom živote sa už určite vyskytol veľakrát, rôznym spôsobom zabalený. Najprv to bolo v období dospievania, keď nás viedli naši rodičia a škola, formou rôznych zákazov a príkazov, neskôr vo forme samostatne získaných vedomostí pri zapájaní sa do života spoločnosti (literatúra, film, médiá, ľudia). Tieto obdobia sú poznačené poznávaním a objavovaním etických princípov uznávaných v spoločnosti, do ktorej sme prišli z vyššej moci. A majú spoločné ešte niečo: vo veľkej miere sme mali malú možnosťou rozhodovať sa a kreovať tieto pravidlá z vlastnej vôle. Takpovediac nám boli vnútené. Nie je mojím cieľom hodnotiť správnosť tohto postupu, beriem ho na tomto mieste ako fakt.

Hodnoty

Obdobie, v ktorom sa nachádzame teraz a obdobie nasledujúce sa vyznačuje, ako som už naznačil, vysokým stupňom voľnosti pre realizáciu našich vnútorných zámerov, pre smerovanie našej osobnosti v ďalšom živote. Toto obdobie so sebou prináša, okrem iného, aj vytváranie a uplatňovanie nových, vlastných etických princípov. Takéto počínanie je prirodzené a prirodzené je, že popri ňom vznikajú mnohé riziká. Podľa akých vzorov vzniknú tieto „nové“ etické hodnoty? Ako ovplyvnia okolie ich nositeľa? Za najväčšie riziko takýchto hodnôt považujem vplyv a moc ich nositeľa na svoje okolie. Ten sa dostáva do pravidelného kontaktu s ľuďmi najmä vo svojom zamestnaní, kde sa jeho pôsobenie prejavuje. Ak za nositeľa týchto hodnôt považujeme absolventa FEI STU, softvérového inžiniera, člena tímu, tak je tento prejav ešte výraznejší, keďže sa predpokladá jeho úzka spolupráca s členmi tímu na konkrétnom projekte. Na mieste

je teraz otázka: Je prirodzené a etické opísaný proces vzniku „nových“ etických princípov regulovať? Ak uvážime, že naša spoločnosť uznáva edukačné spôsoby regulácie svojich členov, tak áno. Rozhodujúcu úlohu v tomto procese zohráva človek, alebo skupina ľudí, ktorí sú pre „tvorcu“ vzorom, či už prirodzeným alebo určeným. V prípade určeného vzoru mám na mysli niekoho, koho si nevyberá, alebo nevyberá sám - najmä vodcu tímu, softvérového manažéra alebo šéfa firmy, v ktorej je zamestnaný. Takýto líder má na svojich pleciach etickú zodpovednosť viesť ostatných k taktiež etickému správaniu. Vzhľadom na svoje postavenie a túto úlohu musí ísť o jedinca zvlášť eticky vyspelého, zrelého, schopného motivovať ostatných. Jeho úlohou je viesť ostatných svojím príkladom, činmi a najmä konzistenciou hovoreného a konaného. Jeho etický kódex, ktorým sa riadi, je verejne známy, čím dáva najavo svoju otvorenosť a vôľu byť kontrolovaný ostatnými. Svoj etický kódex, vlastný alebo odvodený, propaguje etickým spôsobom. Stavia ho na určitých princípoch, overených časom. Aké sú to princípy? Čím sa vyznačujú, že sú vhodné pre prax softvérového manažéra? Sú to také princípy, ktorých aplikácia je adaptabilná v čase, tzn. prirodzene reaguje na meniace, vyvíjajúce sa prostredie, pričom platnosť princípov samotných ostáva nezmenená. Jedna množina takýchto princípov, vhodná pre použitie v oblasti informatiky ale aj inej, sa nazýva čestné informačné princípy [3]. Skladá sa z piatich hodnôt.

Hodnota prvá: partnerstvo. Každý, s kým prídeme pri realizácii projektu do kontaktu, je pre nás partnerom. Nielen kolegovia, nadriadení, podriadení pracovníci, ale aj zákazník.

Hodnota druhá: spravodlivosť. Skladá sa z viacerých konceptov. Otvorenosť, s akou získavame a používame informácie. Znamená aj, že informácie poskytujeme iba tým, ktorí ich použijú legálne, eticky a zodpovedne. Taktiež znamená, že odmietame objednávky, ktoré nezodpovedajú našim etickým štandardom a overujeme si informácie, ktoré získavame, snažíme sa o ich presnosť a správnosť.

Hodnota tretia: rovnováha. Vytvárame rovnováhu medzi tým, čo dávame a čo dostávame. Každý produkt alebo služba, ktorú ponúkame, musí priniesť partnerovi úžitok, väčší ako riziko straty.

Hodnota štvrtá: vzdelávanie. Vzdelávame sa nielen my, ale vzdelávame aj svojich partnerov. Učíme ich, čo robíme a ako to robíme. Ako narábame s informáciami, ktoré nám poskytujú a pomáhame im vyberať si a ovplyvňovať to, čo pre nich robíme.

Hodnota piata: dialóg. Je výsledkom predchádzajúcich hodnôt. Znamená, že počúvame svojich partnerov, reagujeme na ich slová otvorene, bez obrnenia sa, spôsobom, ktorý víta ich snahu pomôcť nám.

Použitie takýchto hodnôt je lepší spôsob vedenia k etickému správaniu, ako stanovenie množiny pravidiel a obmedzení. Hodnoty ako také nie sú obmedzujúce. Nabádajú k samostatnému-tvorivému napĺňaniu svojho obsahu. Tvorivá činnosť, človeku vlastná, sa stáva jeho individuálnou, pozitívnou realizáciou týchto hodnôt. Zákazy a regulácie vedú často k ich úzkostlivému sledovaniu a dodržiavaniu, bez možnosti vlastného prínosu. A čo potom zostane, keď vezmeme človeku tvorivosť!?

Záver

Popri všetkých etických pravidlách, hodnotách a zásadách je dôležité, aby softvérový inžinier vedel rozoznať, že v danej situácii vznikla etická dilema. Toto nie je vždy ľahko rozoznateľné, pretože to čo je legálne (zákonné), ešte nemusí byť etické. Veľmi závisí na cite jeho osobnosti a mikropohľadu na danú situáciu.

Dodržiavanie etických hodnôt závisí výlučne od nášho slobodného rozhodnutia. Naše morálne kvality môžeme zlepšovať pasívne, nevedomým kopírovaním prirodzených alebo určených vzorov, alebo aktívne, vedomým konaním, vzdelávaním sa s cieľom dosiahnuť vyššie etické hodnoty. Takéto konanie je potom zdrojom nášho budúceho zisku. Nie iba materiálneho, predovšetkým.

Použitá literatúra

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

[26] Bodi, Sonia. Ethics and information technology: Some principles to guide students. Journal of Academic Librarianship, Nov98, Vol. 24 Issue 6, pp. 459-464

[27] Van Skilling, D. Values, ethics and data about people. Vital Speeches of the Day, 8/15/96, Vol. 62 Issue 21, pp. 659-662

Bližšia košeľa ako kabát

Roman Ďumbier

Profesia softvérového inžiniera je pomerne nová a preto veľmi nepoznaná. Aby však bola uznaná verejnosťou musí mať črty, ktoré majú aj ostatné, dlho zavedené profesie. K tomu má dopomôcť aj Etický kódex softvérového inžiniera [1]. O jeho dodržiavaní a jeho vplyve na profesiu pojednáva aj táto esej reagujúca na jeho princíp 6 – Profesia.

Úvod

Prvé kódexy etiky pre inžinierov začali vznikať začiatkom 20. storočia v rozvinutých krajinách ako USA, Nemecko a pod. Tieto dokumenty neobsahovali presný opis etických noriem, ktorými sa mali inžinieri riadiť, ale udávali dobrý smer, ktorým sa bolo potrebné uberať. Vznikali v dôsledku potreby začleniť inžinierov, odborníkov, profesionálov do sociálneho a spoločenského povedomia. Táto tendencia sa ďalej prehlbovala s ďalším vývojom techniky v 50-tych a 60-tych rokoch. Medzi pôvodné spoločensky uznávané profesie (lekári, učitelia, atď.) sa začínali dostávať aj prvé technické disciplíny [2].

V súčasnosti je jednou z najrýchlejšie sa rozvíjajúcich disciplín softvérové inžinierstvo. Zo svojich prvopočiatkov, kedy bola tvorba softvéru skôr živelná, bez väčších pravidiel, sa vyvinula na plnohodnotnú technickú disciplínu – profesiu softvérového inžiniera.

Profesia a etika

„Softvéroví inžinieri by mali zlepšovať nepoškvrnenosť a povesť profesie v zhode s verejnými záujmami.“ Toto sú úvodné slová šiesteho princípu kódexu etiky softvérového inžiniera [1]. Mali by sme si však najprv povedať, čo je to etika v softvérovom inžinierstve a čo teda znamená správať sa eticky.

Použil by som slová Donalda Gotterbarna [3], ktorý hovorí: „Myslím si, že etiku softvérového inžiniera, ako profesijnú etiku, možno rozdeliť do troch rôznych úrovní

etiky. Softvéroví inžinieri musia mať tú istú minimálnu morálku, ako zvyšok populácie, teda neklamať, nekradnúť, neubližovať iným atď.“

Tento **prvý stupeň** etiky pozostáva z množiny etických hodnôt, ktoré sú rozšírené medzi nami všetkými tým, že žijeme v určitej spoločnosti.

Druhý stupeň etiky je spoločný pre všetkých profesionálov, ktorý sa starajú o ľudí ovplyvnených ich pracovnou činnosťou.

Tretí stupeň etiky zahrňuje tie povinnosti, ktoré vyplývajú zo špecifickej oblasti akou softvérové inžinierstvo je.

Čo znamenajú rozdiely medzi týmito stupňami? Znamenajú, že profesionálna etika softvérového inžiniera je: ako ostatné etiky požadujúce fair-play (úroveň 1), ako ostatné profesionálne etiky, ktoré sú ovplyvnené špecifikami tej ktorej profesie (úroveň 2) a líši sa od ostatných etík pri uplatňovaní špecifických vzťahov vyplývajúcich z povahy softvérového inžinierstva (úroveň 3).

Etika softvérového inžinierstva ovplyvňuje každé rozhodnutie softvérového inžiniera, ktoré urobí počas celého procesu tvorby softvéru, teda v jeho návrhu, implementácii, testovaní a údržbe.

Princíp 6: Profesia

Prostredie ovplyvňuje človeka

Bod 1.: Pomáhať vytvárať pracovné prostredie napomáhajúce k etickému správaniu.

Etika a morálka vo všeobecnosti zasahuje do činnosti všetkých ľudí, ktorí sa ňou riadia. Podobne to platí aj pri etickom kódexe softvérového inžiniera. Je však veľmi ťažké dodržiavať pravidlá etiky v nevhodnom prostredí. Ak na softvérového inžiniera tlačí zamestnávateľ, nadriadený a pravidlá nedodržiavajú ani ostatní pracovníci, dodržiavanie etických pravidiel mu môže spôsobiť veľké nepríjemnosti. Príkladom môže byť uvedenie nesprávneho odhadu nákladov a určenia ceny tvoreného produktu na príkaz nadriadeného, čím sa firme podarí získať zákazku. Na rozdiel od toho, ak zamestnávateľ akceptuje, prípadne podporuje, etické správanie, a to aj so všetkými jeho nevýhodami, je aj pre softvérového inžiniera ľahšie prispôsobiť sa. K vytvoreniu vhodného prostredia však nestačí len príkaz zhora, je potrebný aj aktívny prístup všetkých zainteresovaných. Každý softvérový inžinier by preto mal začať pomáhať vytvárať etické prostredie od seba. Len keď bude pravidlá dodržiavať veľká väčšina softvérových inžinierov, bude profesia softvérového inžiniera uznávaná.

Profesia a verejnosť

Bod 2.: Zvyšovať povedomie verejnosti o profesii softvérového inžiniera.

Mnoho ľudí si softvérových inžinierov predstavuje ako osoby pripomínajúce šialených vedcov typu Frankeinsteina. Predstavujú si ako ťukajú do počítačov nezrozumiteľné formulky a tešia sa nezrozumiteľným výpisom. Myslia si o nich, že von vychádzajú len v noci a boja sa ľudí.

Takúto predstavu o softvérových inžinieroch spôsobila počiatočná fáza tvorby softvéru, keď verejnosť nemala prístup k počítačom, tie sa nachádzali iba vo výskumných strediskách, vojenských laboratóriách a pod. Softvéroví inžinieri nemali kontakt s verejnosťou, a preto o nich a o ich práci vznikli rozličné príbehy. K tejto predstave v poslednom čase pridávajú aj niektoré, nie celkom reálne, filmy o hackeroch.

Úlohou softvérových inžinierov je teda zmeniť, prípadne vytvoriť správnu predstavu o profesii softvérový inžinier. Mali by predstaviť svoju prácu, jej výsledky, zdôrazňovať kladné stránky profesie, prípadne diskutovať o záporných. Ako pomôcť zvýrazniť profesiu hovorí aj ďalší bol princípu.

Bod 3.: Rozširovať povedomie verejnosti o softvérovom inžinierstve príslušným členstvom v profesijných organizáciách, účasťou na stretnutiach a vydávaním publikácií.

Rozširovať povedomie o profesii softvérového inžiniera môžeme sami vo svojom okolí. Ak však chceme, aby bola profesia naozaj vážená, musíme ju prezentovať aj širokej verejnosti.

Byť vzorom pre ostatných

Bod 4.: Podporovať, ako súčasť profesie, ostatných softvérových inžinierov snažiacich sa o dodržiavanie tohoto kódexu.

Mladí ľudia prichádzajúci do praxe sú svojou prácou často veľmi nadšený a prahnú po úspechu. Môže sa im ľahko stať, že ich konanie nebude etické. Je preto našou úlohou viesť ich a pomáhať im správať sa eticky. Môžeme to urobiť radou, pomocou v probléme, kedy by bolo napríklad ľahšie použiť prácu iného, prípadne uviesť nesprávne údaje. Musíme mu teda ukázať, že je lepšie akceptovať neúspech (dielčí, či aj celkový), ako porušiť etický kódex. Jeho porušením by sme totiž mohli prísť o viac.

Ja už na poslednom mieste

Bod 5.: Nepovyšovať vlastné záujmy na úkor profesie, klientov alebo zamestnávateľa.

Každý softvérový inžinier je človek a teda mu pri práci ide o jeho základné potreby. Chce mať prácu, aby dostával mzdu a mal z čoho žiť. Vecou etiky je však určiť, čo môže urobiť, aby dosiahol svoj cieľ. Softvérový inžinier závisí od svojho zamestnávateľa a ten od svojich klientov. Niekedy je teda veľmi ťažké, zachovať sa eticky. Je veľmi individuálne, čo sa je človek ochotný zriecť preto, aby sa zachoval eticky.

Ďalším problémom môže byť rozhodnutie v prípade, keď na zamestnanca tlačí jeho zamestnávateľ. V niektorých krajinách (napr.: v Japonsku) sa lojalita zamestnanca k zamestnávateľovi veľmi cení [5]. Táto lojalita môže pritom byť väčšia ako etický vzťah k profesii. (Pozn.: V niektorých firmách v Japonsku, lojálni pracovníci po

dovŕšení určitého veku, majú zabezpečenú prácu až do dôchodku, bez ohľadu na ich prípadnú zníženú výkonnosť.)

Verejnosť nadovšetko

Bod 6.: Dodržiavať všetky pravidlá týkajúce sa práce softvérových inžinierov, okrem prípadov, keď takýto postoj nie je v zhode s verejnými záujmami.

V podstate pre každú ľudskú činnosť sú určené nejaké pravidlá. Tie pritom môžu byť určené príkazmi, zákazmi, povoleniami a pod. Takýmito pravidlami sa riadi aj práca softvérových inžinierov. Či už ide o zákon, o pravidlá zamestnávateľa, prípadne iné pravidlá, je potrebné ich dodržiavať. Len týmto spôsobom je totiž možné zabezpečiť kvalitnú a bezpečnú prácu. Ani toto pravidlo však neplatí absolútne. V určitých špecifických prípadoch, môže byť dodržanie, v tom prípade nezmyselného, pravidla nevhodné. Môže ísť napríklad o tvorbu medicínskeho softvéru, kde štandardné postupy nemusia stačiť na zabezpečenie bezpečnosti pacientov, prípadne lekárov. Je teda na softvérovom inžinierovi, aby zvážil dôsledky svojich rozhodnutí a podľa toho sa rozhodoval.

Nepodvádzaj, nikto nie je dokonalý

Bod 7.: Byť presný pri stanovovaní vlastností softvéru, na ktorom pracujú a vyhnúť sa tým stanoveniu nielen nepravdivých požiadaviek, ale aj požiadaviek špekulatívnych, klamlivých, pochybných a nič nehovoriacich.

Pri stanovovaní vlastností vychádzajú softvéroví inžinieri so svojich skúseností a znalostí. Mali by sa snažiť vlastnosti stanovovať čo najpresnejšie, aby bola použiteľnosť softvéru čo najväčšia. Je úplne samozrejmé, že by pri ich stanovovaní nemali klamať. Nemali by však ani hovoriť o vlastnostiach nejasne a nezrozumiteľne a tým sa snažiť získavať dôveru a uznanie. To je možné získať len vhodnou prácou na zverenom projekte.

Bod 8.: Preberať zodpovednosť za hľadanie a opravu chýb, za vytváranie správ o chybách v softvéri a v príslušných dokumentoch, na ktorých pracujú.

Softvéroví inžinieri, sú ako aj ostatní, ľudia omylní. Nemôžeme sa teda čudovať, že počas svojej práce, ktorá je navyše zväčša tvorivá, urobia nejaké chyby. Je ale dôležité, aby si tieto chyby vedeli softvéroví inžinieri priznať a zobrať za ne zodpovednosť. Pričom je dôležité opravovať chyby nielen v samotnom softvéri, ale aj v príslušnej dokumentácii. Iba takto možno zamedziť problémom pri ďalšom vývoji nimi vytvoreného softvéru. Aj keď je uznanie chyby často nepríjemné, môžeme si ním získať úctu a dôveru ostatných, a to klientov, spolupracovníkov ako aj zamestnávateľa.

Informuj, kým nebude neskoro

Bod 9.: Zaistiť, aby klienti, zamestnávatelia a vedúci pracovníci vedeli o záväzku softvérového inžiniera dodržiavať tento kódex a z toho vyplývajúcich ostatných záväzkov.

Ak chce softvérový inžinier dodržiavať určité pravidlá, mal by s nimi oboznámiť svoje okolie. Ináč by sa mu mohlo stať, že od neho budú požadovať niečo, čo nezodpovedá jeho presvedčeniu. V takejto vyhranenej situácií už býva neskoro oháňať sa etickým kódexom, často by to bolo považované za zlyhanie. Je preto vhodné svoje okolie s rozhodnutím dodržiavať etický kódex dopredu oboznámiť. V takom prípade je odmietnutie niečoho neetického akceptovateľné. A navyše je oboznámenie držania sa človeka určitého kódexu aj zárukou pre prípadných klientov.

S ním nemám nič

Bod 10.: Vyhýbať sa spájaniu s ľuďmi, spoločnosťami a organizáciami, ktoré porušujú tento kódex.

Nie každý softvérový inžinier a organizácia je stotožnená s pravidlami etického kódexu. Ich hodnoty môžu byť dosť odlišné, dosiahnuť zisk za každú cenu býva často príťažlivé. Takýto ľudia však poškodzujú povesť profesie a môžu poškodiť aj našu povesť. Ak teda nechceme, aby nás s nimi spájali, musíme to dať patrične najavo, prípadne vysvetliť, že je medzi nami rozdiel. Treba zdôrazniť, že sa od nich dištancujeme práve preto, že nedodržiavajú kódex etiky. Tým sa od nich odlíšime a ukážeme, že sa to dá robiť aj ináč.

Softvérový inžinier

Bod 11.: Akceptovať, že porušovanie tohoto kódexu nie je v zhode so správaním sa profesionálneho softvérového inžiniera.

ACM a IEEE Computer Society, najväčšie organizácie združujúce softvérových inžinierov, vydali tento kódex etiky v presvedčení, že sa ním budú riadiť nielen všetci členovia týchto organizácií, ale všetci, ktorí sa za softvérových inžinierov považujú. Preto, ako príslušníci profesie softvérové inžinierstvo, máme za povinnosť dodržiavať tento kódex. A preto nikto, kto tento kódex porušuje, si nemôže hovoriť softvérový inžinier.

Keď to už ináč nejde

Bod 12.: Vyjadriť nespokojnosť ľuďom zúčastneným na závažnom porušení tohoto kódexu, keď to nie je nemožné, kontraproduktívne alebo nebezpečné.

Bod 13.: Oznámiť podstatné porušenia tohoto kódexu príslušným autoritám, keď je jasné, že konzultácie s dotknutými osobami sú nemožné, kontraproduktívne alebo nebezpečné.

Väčšina softvérových inžinierov sa počas svojej praxe stretne s kolegom, ktorý nedodržiava kódex etiky softvérového inžiniera. Ako by sa mal zachovať? Mal by sa len nečinne prizerať? Nie, lebo jeho kolega svojou činnosťou poškodzuje všetkých softvérových inžinierov, teda aj jeho. Mal by sa teda pokúsiť zmeniť myslenie a tým aj konanie svojho kolegu. Najlepšie ak to urobí tým, že mu pôjde príkladom a ukáže mu cestu. Ak však náprava nie je možná je potrebné porušovanie oznámiť príslušným nadriadeným orgánom, aby mohli zaujať stanovisko a vyvodiť dôsledky.

Záver

Profesia softvérového inžiniera je veľmi mladá, dynamicky sa vyvíjajúca profesia. Jej súčasťou sú ľudia z mnohých krajín sveta s rozdielnou mentalitou, kultúrou, sociálnou a spoločenskou situáciou. Etický kódex je pokusom o výber toho najlepšieho a najvhodnejšieho, čím sa má softvérový inžinier riadiť pri tvorbe softvéru tak, aby pritom nepoškodzoval česť profesie. Ide teda o to, akým spôsobom má jednať so svojím okolím, teda s spolupracovníkmi, zamestnávateľmi a klientmi a akým spôsobom sa má stavať k svojej práci. Verím, že tento etický kódex sa stane základom pre všetkých softvérových inžinierov u nás, aj vo svete.

Použitá literatúra:

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

[29] Christelle Didier, Why there are no Engineering Ethics in France: an historical interpretation, Centre d'Ethique Contemporaine, Université Catholique de Lille, 1999, http://www.onlineethics.com/essays/didier.html

[30] Donald Gotterbarn, Forum for Advancing Software engineering Education (FASE), Volume 8 Number 12 (107th Issue) - December 15, 1998, http://www.cs.ttu.edu/fase/v8n12.txt

[31] Bushido, Japonská etika bojovníkov, Prednáška z predmetu: Manažérska etika, prednášateľ: doc. Eva Sládková (šk. r.: 1999/2000)

Kolegovia – problémy a riešenia

Tomáš Milička

Esej rozpracováva siedmy princíp Etického kódexu softvérových inžinierov [1], pojednávajúci o kolegoch a dodržiavaní určitých etických pravidiel s tým súvisiacich. Princíp Kolegovia má osem bodov. Ku každému najprv vyjadrujem určitý všeobecný postoj, tento objasňujem, uvádzam pozitívny alebo negatívny príklad a na záver navrhujem nové alebo už v praxi overené riešenie. V jednotlivých bodoch ide predovšetkým o dôveru v prácu iných, jej objektívne a úprimné posúdenie, vypočutie a rešpektovanie názorov kolegov, pomoc pri zabezpečení informácií, korektnom správaní k ostatným, ako i uvedomenie si svojich kompetencií.

Úvod

Odpoveď na otázku, ako sa chovať v práci, sa zdá byť rovnako jednoduchá, ako aj historicky overená. Dobre pracovať a nič okolo si nevšímať. Platí ešte stále táto stará „múdrosť“ našich dedov? Zdá sa, že v modernom a hlavne „supermodernom“ svete informačných technológií to už asi nestačí. Práve v tejto spoločnosti, platí množstvo nových zásad, ktorých princípom je závislosť úspechu pracovníkov od tímového prístupu.

7.01 Nabádať kolegov aby sa pridržiavali tohto kódexu

Aby sa kódex dostal do povedomia softvérových inžinierov, je potrebná neustála osveta medzi nimi. Základom masového rozšírenia je správanie sa vedúcich pracovníkov v duchu zásad stanovených týmto kódexom. Každý vedúci pracovník by mal byť pre svojich podriadených vzorom. Takýmto prístupom úloha šíriteľov prejde i

na pracovníkov - kolegov, čím sa postupom času stane dodržiavanie určitých zásad samozrejmosťou. Príkladné správanie kolegov bude príkladom pre ďalších.

Jedným z riešení je priame zviditeľnenie základných princípov na pracovisku (plagát na stene ironizujúci jednotlivé princípy), aby zamestnanci prichádzali s nimi tak často do kontaktu, aby sa stali samozrejmosťou.

7.02 Napomáhať kolegom v profesionálnom raste

Táto otázka je dosť citlivá, lebo nie všetci sú ochotní zdieľať svoje ťažko nadobudnuté vedomosti alebo zložito vyhľadané informácie s ostatnými, keďže mu tieto znalosti prinášajú profesionálne výhody. Táto neochota je spôsobená obavami o stratu profesionálnej jedinečnosti a nenahraditeľnosti. Prekonanie tohto strachu vyžaduje hlbokú schopnosť tímového myslenia každého softvérového inžiniera. Dôležitá je vysoká odborná úroveň tímu ako celku, nielen jeho jednotlivcov.

Riešením by mohli byť napr. koncotýždňové stretnutia kolegov na neformálnej úrovni, kde by každý zhrnul s čím novým sa počas týždňa stretol. Či už by išlo o skúsenosti získané z kníh, z Internetu alebo zo stretnutí s inými kolegami. Dôsledkom komunikácie na takejto úrovni by mala byť výmena informácií a skúseností pri riešení nových problémov, s ktorými sa stretli [3].

7.03 Plne dôverovať práci ostatných a zdržať sa pripisovania nadmerných zásluh

7.04 Posudzovať prácu iných objektívne, úprimne a náležite zdokumentovaným spôsobom

Aby sme dosiahli pozitívny výsledok pri každej zadanej úlohe, je nutná vzájomná dôvera všetkých členov pracovného tímu. Prenesenie zodpovednosti na podriadených pracovníkov alebo kolegov v nich vyvoláva pocit spolupatričnosti a motivuje ich. Dôvera v pozitívny výsledok ich spoločného úsilia je cestou k objektívnemu a úprimnému posúdeniu ich výsledkov. Pri tímovej práci spôsobuje nedôvera v schopnosti ostatných stratu prostriedkov a času a dochádza i k zhoršeniu vzťahov na pracovisku. Pri nedôvere pracovník napríklad sústredí svoju pozornosť na kolegov problém, len aby dokázal jedinečnosť svojich schopností.

Riešením môže byť ponechanie voľného priestoru pracovníkovi, ktorý tým na základe pracovných výsledkov presvedčí ostatných o svojej dôveryhodnosti. V dôsledku toho bude jeho práca posudzovaná objektívne a úprimne.

Ďalším riešením by mohlo byť presné rozdelenie úloh medzi kolegov vedúcim tímu, pričom by boli určené jednoznačné kompetencie. Rozhodnutie šéfa by malo všetkých presvedčiť, že pracovník vybraný na túto čiastkovú úlohu je ten správny.

Ako ideálny príklad poslúži firma Intel, ktorá v súčasnej dobe produkuje 90% mikroprocesorov do osobných počítačov a jej kapitálová hodnota predstavuje 114 miliárd dolárov, čo bezpochyby svedčí o tom, že patrí k najlepším.

Zamestnanci sú tu za svoju prácu zodpovední nielen svojmu nadriadenému ale aj svojím kolegom. Každý pracovník musí byť schopný kedykoľvek predložiť výkaz, v ktorom veľmi stručne vlastnými slovami vyjadruje, na čom práve pracoval, ako sám seba hodnotí, čo hodlá robiť ďalej a ako sa chce vyrovnať z prekážkami. Najzaujímavejším momentom tejto akcie je určite fakt, že výkaz neslúži pre žiadnu kontrolu zo strany vedenia firmy, ale je predkladaný iba spolupracovníkom. Takýmto konaním sa dosiahne dôvera v prácu iných, ako i jej objektívne a úprimné posúdenie [2].

7.05 Vypočuť názory, obavy alebo sťažnosti kolegov

Tento princíp by mal byť základnou črtou vzťahov vo firme. Tímová práca predpokladá neustály kontakt jednotlivých členov kolektívu. Každý člen má svoje problémy, pracovné i osobné, a hľadá v kolektíve niekoho, komu sa môže zveriť. Tu je nevyhnutný určitý stupeň empatie všetkých kolegov. Spolu môžu jednoduchšie nájsť pre všetky problémy vhodné riešenie.

Opäť je tu príklad zo spoločnosti Intel, ale určite by sme podobný našli aj v mnohých iných spoločnostiach vedúcich súčasný trh informačných technológií. Vedenie spoločnosti trvá na tom aby každý zamestnanec videl svojho nadriadeného i kolegov pri práci a mohol k nim kedykoľvek vstúpiť, čo platí aj o riaditeľoch. Zmyslom toho všetkého nie je žiadny „okázale humánny“ prístup, ale snaha prelomiť bariéry v osobných vzťahoch medzi kolegami, pretože rozhovor s ľuďmi z očí do očí bez papierových hlásení, vzájomná výmena informácií a možnosť radiť sa za pochodu, je najlepším prostriedkom k uvoľneniu rezerv energie firmy [2].

7.06 Napomáhať kolegom k uvedomeniu si súčasných štandardných praktík vrátane bezpečnostnej politiky a procedúr na ochranu hesiel, súborov a ostatných citlivých informácií a bezpečnostné opatrenia všeobecne

Bezpečnostné opatrenia sa stávajú čoraz dôležitejším aspektom v živote ľudí a dvojnásobne to platí o softvérových inžinieroch. Ochrana svojho duševného vlastníctva, či už kódu programu, návrhu produktu, výsledkov analýz alebo iných citlivých informácií sa stáva každodennou rutinou. Mnoho pracovníkov si stále nevie predstaviť hodnotu týchto informácií a často ju zisťujú až po ich odcudzení alebo poškodení. Pod bezpečnosťou si nepredstavujeme len zabezpečenie prístupu heslom ale, i fyzické zabezpečenie údajov, či už vytváraním záložných súborov, zabezpečenie nepretržitého napájania alebo zdvojením kritických prvkov systému.

Riešením je striktné dodržiavanie bezpečnostnej politiky firmy ako i napomáhanie pri jej vylepšovaní. Pri zistení nedostatkov v zabezpečení vlastníctva kolegov alebo firmy ako entity, by malo byť v záujme každého pracovníka upozorniť na ne a prípadne aktívne napomáhať pri ich odstránení.

7.07 Nepodnikať žiadne nekorektné kroky v kariére ostatných kolegov, i keby obavy zamestnávateľa, klienta alebo záujem verejnosti nútili softvérových inžinierov, v dobrej viere, zapochybovať o schopnostiach kolegov

Pri tímovej práci je nemysliteľné pochybovať o práci kolegov alebo neoprávnene zasahovať do ich kompetencií. Takéto narušenie v silne konkurenčnom prostredí i keď vykonávané s cieľom dosiahnuť pozitívny výsledok, môže naštrbiť vzťahy vo vnútri firmy, ale aj navonok, porušením jej imidžu pred zákazníkmi.

Pri tomto bode nie je jednoduché navrhnúť jednoznačné riešenie. Predchádzanie takýmto problémom je otázkou výchovy, charakteru a individuálnych hodnôt jednotlivca. Pri dodržiavaní zásad tímovosti a kolegiality by k takémuto konaniu v podstate nemalo dôjsť.

Ak už však škoda nastala, riešením je jedine čas. Ak niekto predsa zapochyboval, i keď s dobrým úmyslom, o vašich schopnostiach, časom sa v obore, akým je softvérové inžinierstvo, schopnosť alebo neschopnosť určite prejaví. Výsledkom bude znovunadobudnutie dôvery a jej posilnenie alebo v opačnom prípade jej úplná strata.

7.08 V situáciách mimo svojho poľa pôsobenia, požiadať o pomoc odborníka, ktorý má kompetencie v týchto oblastiach

Pre každého softvérového inžiniera by malo byť samozrejmosťou chápať nutnosť určitého stupňa odbornosti pri riešení problémov v daných oblastiach. Pri neistote alebo neznalosti by sa mal každý obrátiť na kolegu, odborníka v danej problematike. Je nesprávne báť sa preukázať nevedomosti v danej oblasti a to či už vo vzťahu kolega – kolega alebo vedúci – podriadený.

Riešením je opäť jedine vytvorenie pracovného ovzdušia, v ktorom dochádza k bezproblémovej a otvorenej komunikácii (ako bolo už spomenuté pri bode 5).

Záver

Skúsení softvéroví inžinieri, pracujúci neustále v rôznych tímoch, dospievajú k poznaniu, že nejaké zvláštne tajomstvo spolupráce vlastne neexistuje. Jedna taká osvedčená rada hovorí, že sa má pracovník, a nielen softvérový inžinier, chovať jednoducho ako človek s veľkým Č. Inými slovami, ako príjemná bytosť, aby ľudia, od

ktorých sa chce učiť, chceli byť v jeho blízkosti. Mal by sa teda chovať tak, aby sa tí, ktorí mu zverujú úlohy, alebo s ním na týchto úlohách participujú, mohli spoľahnúť, že sa so zadaním vysporiada nielen v duchu svojich základných povinností, ale že k tomu dodá pridanú hodnotu „svojho ja“. Teda pridá k spoločnému riešeniu niečo zo svojich vedomostí, schopností a skúseností. Nejde o žiadne lacné zviditeľňovanie, ale o to byť osobnosťou, aj keď len zavŕtanou do svojho počítača niekde v bunke veľkého oddelenia. A z takéhoto modelu správania sa vyplývajú určité pravidlá – morálny kódex.

Použitá literatúra

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

[33] I. Brož, Tajemství úspěchu = zdravý rozum ?. Computerworld č.3, IDG Publishing, Praha, 1999, str. 12.

[34] J. McCarthy, Softvérové projekty. Computer Press, Praha, 1999.

[35] ACM Code of Ethics and Professional Conduct, 16.1.1998, http://www.acm.org/constitution/code.html.

[36] M. C. Loui, Fieldwork and Cooperative Learning in Professional Ethics, 16.1.1999, http://onlineethics.org/essays/loui.html.

Dračie srdce

Jaroslav Hanúsek

Etika sa stala neoddeliteľnou súčasťou každodenného ľudského správania sa. Softveroví inžinieri, ako vysoko profesionalizovaní a väčšinou úzko zameraní odborníci, často zabúdajú, že aj oni sú ľudia, žijú medzi ľuďmi a podľa toho by sa mali aj správať. Cesta k úspechom je uzučký chodníček, ktorý človek – individualista len veľmi ťažko prekoná sám. Musí sa preto rozhodnúť – buď bude osamotene (ale s hrdosťou) budovať svoju prestíž alebo sa podvolí tímovému duchu a spoločne s ostatnými bude pomaličky napredovať. Táto esej prináša pohľad z jedného mladého softvérového tímu, kde dilema prestíže a kolegiality zohrala veľkú rolu v úspechoch a smerovaní mladých ľudí.

Úvod

História nášho tímu sa začala písať pred deviatimi rokmi. Slová etika a morálka sme poznali iba ak z vyučovania občianskej náuky. Vedeli sme však, že ak chceme s nejakým softvérovým projektom preraziť – jednotlivo to nezvládneme. Každý z nás už mal aké také skúsenosti s vlastnými projektmi a každý z nás si uvedomoval, že viac hláv pravdepodobne dosiahne viac ako jedna. A vtedy začalo obdobie, keď sme pomaly začali zisťovať, čo je to boj o prestíž, snaha presadiť sa, dosiahnuť cieľ a zvíťaziť. Zlé na tom bolo len jedno – nerobili sme to medzi skupinami, ale len v rámci nášho tímu. Začal sa totiž boj o prestíž.

Od začiatku sme vedeli, že začatím práce v tíme strácame kus svojho ega. Bolo nám jasné, že musíme vystupovať ako jeden celok a prezentovať spoločný názor. Náš prvý etický problém sa objavil veľmi rýchlo – zodpovednosť voči cudziemu vlastníctvu. Programátor sa často dostane do obrovskej dilemy – *zodpovednosť voči sebe a individuálne plány postavené proti záujmom ostatných (verejnosti, nadriadených, podriadených)*[4]. Zabudli sme však na jednu vec – zodpovednosť jeden voči druhému.

Začali sme bojovať sami proti sebe a neuvedomujúc si následky, pracovali sme na pomalom rozpade nášho tímu.

Ak sme boli my tí, čo sme išli a ideme za prestížou, neuvedomujeme si, že spolu s nami idú aj naše vzájomné vzťahy. V tímovej spolupráci každý zodpovedá nielen sám za seba, ale aj za všetkých ostatných. Predstaveme si teraz na chvíľku, že sme rytieri z rozprávky o Dračom srdci. Náš drak – tímová etika – nosí naše srdce a my , neuvedomujúc si to – v snahe dosiahnuť vytúžený cieľ – osobný úspech, podvedome bojujeme proti sebe. Bojujeme s Dračím srdcom proti sedemhlavému drakovi.

HLAVA I. Pomáhaj kolegom v profesionálnom rozvoji

Ak by mal byť človek na niečo hrdý, musel niečo dosiahnuť. Ale čo ak niekto robí niečo podobné ako ja ? A čo ak to dokončí skôr ako ja ? A čo ak zožne úspech on...

Takéto a podobné úvahy sa v našom tíme začali objavovať pravidelne, keď sme sa rozhodli pustiť do niečoho nového. Viacerí sme sa pustili do práce na rovnakom probléme, neuvedomujúc si časové straty a nadbytočnosť práce. Ospravedlňovali sme si to tým, že takto síce urobíme niečo rovnako, ale vznikne množstvo nápadov.

Myšlienka to bola pomerne dobrá, malo to len jeden háčik – prestíž. Ak niekto na niečo výborné prišiel, nechal si to pre seba a pre použitie vo svojich osobných projektoch. Miesto toho, aby sa podelil s kolegami na čo prišiel, radšej ich odporučil na vynikajúcu literatúru, kde si to môžu naštudovať. Navonok všetko fungovalo – skupina bola rada, že našla nový zdroj informácií a jednotlivec bol rád, že má náskok pred ostatnými. Tímová spolupráca teda funguje. Navonok sme teda etický princíp vzájomnej pomoci dodržali. V skutočnosti sme drakovi odsekli prvú hlavu...

HLAVA II. Objektívna kritika

Kritizovať prácu v tíme väčšinou všetkých baví. A klamal by som keby som tvrdil, že takáto kritika je zbytočná – práve naopak. Keď sme si po dokončení nejakej časti programu išli sadnúť do mesta na „pracovnú schôdzku“, nevenovali sme sa hodnoteniu tej, ktorej časti, ba ani hľadaniu, čo by sme mohli robiť ďalej. Všetka konštruktívna kritika sa upriamila na opisovanie chýb a neschopnosti toho druhého. Jedným slovom – konštruktívna kritika sa mení na kritiku osobnú. Ľudí teší, keď ich produkty sú menej chybné a keď tí ostatní majú problémy.

V tomto momente by ešte bolo všetko ako tak v poriadku. Naša kríza prišla však vtedy, keď sme začali hodnotiť rovnocenné produkty. Nie aby sme sa snažili upriamiť na vzniknuté chyby a nedostatky, ale uprednostnili sme umelé vychvaľovanie svojich komponentov. A to už jednoznačne išlo o udržanie prestíže. Nebolo prípustné, aby človek, ktorý sa označoval ako „1st coder“ naprogramoval niečo horšie ako „4th“ programátor. A ak to tak aj náhodou bolo, tak ten, komu išlo o udržanie nepísaného postavenia, najbližšiu noc určite celú svoju časť preprogramoval. Nie preto, aby bol

výsledok tímového projektu na lepšej úrovni, ale aby sám sebe dokázal, že je lepší. Kritika teda splnila svoj význam, či to však zlepšilo efektivitu práce v tíme a lepší etickejší prístup k práci druhého, nie je až také jednoznačné.

HLAVA III. Úcta a pomoc

Porušenie etického princípu úcty sme v tíme porušili asi najmenej. Bolo to totiž TO, čo nás ťahalo dopredu. Úcta nie jeden, k druhému, ale úcta ku kolegom, ktorí to už niekam dotiahli. V tomto ohľade sme vedeli byť veľmi "tímoví". Ak sme išli proti niekomu, teda ak sme chceli dosiahnuť úspechy také, aké kolegovia v zahraničí, museli sme si navzájom pomáhať. Zamerali sme sa na špičku v obore a snažili sme sa vyrobiť konkurencie schopný produkt.

Má to len jeden háčik. Úcta sa veľmi rýchlo mení na závisť a neskôr na averziu. Pokiaľ sme dúfali, že niečo sa nám podarí dotiahnuť do takej podoby ako európska špička *Future crew* (aj keď s niekoľko ročným odstupom), pomáhali sme si, ako to len išlo. Akonáhle sme však zistili, že na niečo nemáme, úcta sa zmenila na nedosiahnuteľný ideál, teda niečo, čo je v praktickom živote úplne nepodstatné. Prešli sme teda z úcty k ignorácii. Pokiaľ sme boli našimi výsledkami presvedčení, že sme prišli na koniec svojich možností, úcta sa vyparila. Jednoducho si nevieme priznať, že tí najlepší si tú úctu naozaj zaslúžia a naša prestíž pri nich končí.

HLAVA IV. Udržanie tajomstva – úcta k dôverným údajom

Etický kódex inžiniera informatika je v tomto princípe neoblomný: *„chráň právo súkromia každého používateľa a správcu údajov“* [3] Mali by sme teda chrániť každú informáciu, či je to už databáza používateľa alebo firemný adresár. Ale dodržujeme si navzájom právo súkromia sami medzi sebou ? Ako vieme, čo ten, ktorý kolega považuje za dôverné ?

Keď už naše staré projekty naštrbil zub času, rozhodol som sa, že niektoré zdrojové kódy by možno pomohli ďalším začínajúcim programátorom. Z etickej stránky to vyznieva veľmi pekne, v praxi to však skončilo úplne odlišne. Všetci členovia tímu totiž nezdieľali môj názor. Ich proti argumentom bol fakt, že na starých programoch pracovali kedysi dávno veľmi dlho a teraz by mal niekto veľmi lacno získať ich prácu. Mojim protiargumentom bol fakt, že išlo o starý kód a v súčasnosti použiteľný už len na edukatívne účely. Hlavným problémom však zostalo, že majiteľom kódov je tím. A nikto presne nepovedal aké práva z toho vyplývajú jednotlivým členom. Nakoniec boli kódy zverejnené, ale za cenu naštrbených vzťahov v tíme. Bol to ďalší krôčik k zničeniu draka.

HLAVA V. Umenie vedieť komunikovať

Počúvať jeden druhého je veľmi ťažké. Ešte ťažšie je pochopiť toho druhého. A najťažšie je pomôcť a zároveň neublížiť. Musíme sa zmieriť s faktom, že tu nie sme sami a chtiac, či nechtiac sa zúčastňujeme *procesu socializácie* [2]. Celý idealizovaný proces socializácie, teda dosiahnutia spoločnosti dodržujúcej etické princípy, však má pred sebou niekoľko bariér. Snažíme sa žiť tak, aby sme sa dostali do lepšej vrstvy, aby sme boli bohatší, aby sme mali úspech. Žiaden z týchto „materialistických" cieľov sa nemá príliš v láske s etickými princípmi vo všeobecnosti. Pri tomto našom napredovaní často zabúdame na ľudí, na ktorých sme závislí.

Vytvárame si vlastné bariéry – nerozprávame sa s ľuďmi, ktorými pohŕdame, s podriadenými, s ľuďmi, ktorých sa bojíme a ani s kolegami, o ktorých si myslíme, že nám nič nemôžu povedať.

„Od mala sme vychovaní žiť tak, aby sme sa bavili len s istými ľuďmi a tváriť sa akoby tí ostatní tam ani neboli, akoby neexistovali. A ak sme už na niečo zvyknutí, zdá sa to normálne, tak nám padne veľmi zaťažko, ešte čosi meniť. Pretože človek nemá zmeny rád a občas má z nich strach".[6]

V tíme sa nám vykryštalizoval jeden programátor, ktorého sme nemohli ináč nazvať, ako človeka, ktorý máva stále nápady a riešenia trošku mimo. (autor v [1] výstižne označuje takýchto ľudí ako „ňoumov"). V tej dobe sme si však neuvedomovali, že aj nápady veľmi vzdialené realite sú niekedy dobré nápady. Tento náš kolega informatik nakoniec z tímu odišiel a my sme si až odstupom času uvedomili, čo sme stratili. A to len preto, že sme nevedeli počúvať. Alebo nechceli...

HLAVA VI. Nechať si poradiť

„*Inžinier by mal sprostredkovať svoje znalosti tým, ktorí to potrebujú*", píše sa v etickom kódexe pre informatikov [3]. Ale mali by sme si položiť aj otázku, či sme ochotní prijať informácie od svojich kolegov. Neprijímame totiž iba informácie, ale aj názory. A s týmto pocitom sa nevieme niektorí zblížiť. Patrilo by sa asi povedať, že v našom tíme je väčšina ľudí, ktorí dostali do vienka vlastnosť tvrdohlavosti.

Ľudia, ktorí majú silné ego, neradi prijímajú názory od druhých. A najhorší prípad je, keď by mala osoba s nižšou prestížou poučovať osobu, ktorá sa cíti aspoň o kúsok nadradenejšia. V tom momente sa totiž aktivuje v ľuďoch akýsi pocit, či pud sebazáchovy. Táto skúsenosť sa nám v bežnom živote prejavila, keď sme sa stretávali s konkurenčnými tímom, v ktorom boli vekovo mladší členovia. A v nás akoby zvíťazilo vlastné sebapresvedčenie, že určite dosiahneme aj vlastnými silami a schopnosťami to, čo nám ponúkal druhý tím – spoluprácu. Nechali sme si teda poradiť, nie však reálnym uvažovaním ale vlastnou tvrdohlavosťou .

Výsledok sťatia tejto hlavy bol jasný. Konkurenčný tím obsadil druhé miesto v celoslovenskej súťaži a my sme sa jej nemohli ani zúčastniť. Nepodarilo sa nám do termínu dokončiť produkt.

HLAVA VII. Neublížiť

Ublížiť niekomu sa dá rôznym spôsobom. Mali sme medzi sebou kolegu, ktorý sa urazil už vtedy, keď sme vo svojich častiach použili rovnaké mená premenných a procedúr ako on. A uraziť (i keď nechtiac) člena tímu môže mať katastrofálne následky, hlavne ak je introvertnej povahy.

V druhom tíme, v ktorom som pracoval v minulosti sa vyskytol prípad, že programátor mesiac pracoval na jednom komponente a po zasadnutí tímu sa rozhodlo, že v systéme tento komponent vôbec nebude. Otázne je, kto spravil chybu. A všeobecne, či vôbec niekto urobil chybu. Vznikla chyba pri plánovaní ? Jedno je však už isté. Tomu programátorovi klesla prestíž, sebaúcta a v svojich očiach videl len mesačnú robotu, ktorá bude odložená a zapadnutá prachom, kým si na ňu niekto nespomenie.

Najväčšiu degradáciu v sebaúcte a prestíži však nesie kritika. A to kritika taká, ktorú si bude dotyčný inžinier považovať za osobnú a nie konštruktívnu. „*Prijímať kritiku nemáme tak, aby nás urazila, ale tak, aby nás prinútila preusporiadať si naše ciele v prospech spoločného cieľa a aby sme si uvedomili myšlienku a nie slová kritiky.*“ [5]

Ak chceme kritizovať niekoho, kto má takéto tendencie, v prípade, že začne zlyhávať – chyba je jednoznačne na našej strane – na strane kritizujúceho. Toto si niektorí ľudia v našom tíme neuvedomili a neuvedomujú ani dodnes. Ublížiť niekomu je určite jednoduchšie ako si ho udobriť. Ak sa v tíme navzájom dostaneme do averzie, nemá vôbec zmysel uvažovať o ďalšej kolegiálnej spolupráci.

Budúcnosť nášho tímu spočívala v budúcnosti jednotlivcov. A to bola tá najväčšia chyba. Všetko sme sa snažili robiť si nezávisle na ostatných a využívať pri tom nejakú abstraktnú idealizovanú formulku vzájomnej spolupráce. Etika vo vzťahoch k druhým mala vážne štrbiny a jednoznačne viedla k rozpadu tímu. Ľudia, ktorí možno (keby sme sa vedeli pozerať okolo seba) nemali ďaleko k dosiahnutiu svojich profesných ideálov, no podľahli vidine prestíže.

Nuž, a prečo tá paralela s Dračím srdcom ? Rytier, ktorý chcel zabiť draka nevedel, že jeho život je srdcom spojený s drakom a to znamenalo - jeden zomrie a musí zomrieť aj druhý. Zabiť draka – odseknúť mu všetkých sedem hláv, znamenalo zabiť sám seba. Zabudnúť na etiku znamená skončiť so spoločnosťou. Uzavrieť sa sám pre seba a konať na úkor druhých. Keď si neuvedomíme, čoho sme súčasťou, budeme vždy bojovať sami proti sebe. Jeden proti všetkým, všetci proti všetkým. Všetci sa budeme hnať za vidinou individuálnej prestíže, vidinou úspechu na úkor druhých. Veď ako vravia v kapitalistických krajinách: „Niekto je hore a niekto musí byť aj dolu“.

Skutočnosť je však desivá - ľudia dodržujúci etické princípy končia väčšinou dole, dole pod nohami niekoľkých karieristov, ktorí ako tvrdia – určujú vývoj.

Každý z nás však má v sebe kúsok Dračieho srdca, každý z nás je zodpovedný nielen za seba, ale aj za ostatných. Žijeme v spoločnosti a to si vyžaduje sociálne správanie, ktorému určuje smer etika a morálka. A aby sa nám podarilo úspešne sa zapojiť medzi ostatných a dokázať, že spolu sa dá ukázať viac – musíme sa ešte veľa učiť. Učiť sa počúvať druhých a nemyslieť len sami na seba.

Použitá literatúra

[37] McCarthy Jim, Softwarové projekty, Computer press, Praha, 1999

[38] OECD, Ethics in the public service, ethics infrastructure, Ocassional papers N#.14 , Paris, 1996, str. 37

[39] Bulletin of the American Society for Information Science, č.25, August/ September 1990

[40] Katz Jon, "Code of Ethics for Programmers?", Tecnology-and-the-Future dept, september 1999, http://shlashdot.org

[41] Texas Instruments, Articles from Ethics and the TIer: Responsibilities as a Team Member, články č. 172, 179 a 183, http://onlineethics.org

[42] Salmova L.,Transparentnosť, morálka a etika v SR,Transparency International Slovensko, Bratislava, Marec 2000, str.50

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

Z vnútra pramení sila

Branislav Šimo

Tento článok si kladie za cieľ podrobnejšie rozobrať myšlienky posledného – ôsmeho – princípu Etického kódexu softvérového inžinierstva, ako bol publikovaný v [1]. Prezentuje dôležitosť jednotlivých bodov kódexu ako návodov na správanie sa počas práce v profesii všeobecne, ako aj v krízových situáciách, kedy je potrebné zachovať si profesionalitu a vydržať tlak okolia v záujme neohrozenia iných.

Úvod

Otázka etiky a etického správania sa čoraz viac objavuje v praxi. Tento jav je snáď najviditeľnejší práve v oblasti softvérového inžinierstva, pretože je to jedna z najmladších inžinierskych disciplín a jej vývoj bol a je značne rýchly, možno povedať živelný. Živelnosť síce môže pohnúť vývoj riadne vpred, môže ukázať a rýchlo prehodnotiť nové cesty, ktoré by sa neobjavili v stojatých vodách, alebo by sa nimi nechcelo nikomu púšťať, má však svoju nevýhodu, a tou je pocit chaosu a nekontrolovateľnosti, ktorý človek stojaci uprostred diania zažíva.

Tento pocit poznajú mnohí pracovníci v oblasti výpočtovej techniky ako aj zákazníci, na ktorých sa stále hrnú nové technológie, informácie, programy, to všetko v rôznej kvalite, s rôznymi alebo žiadnymi zárukami. Pritom výpočtová technika zohráva v našom živote čoraz dôležitejšiu úlohu, a tým rastie aj jej schopnosť spôsobiť dobro alebo zlo.

Etický kódex softvérového inžinierstva si kladie za cieľ do čo najväčšej miery zaručiť, že výsledky softvérového inžinierstva sa použijú na dobro, nespôsobia žiadnu ujmu a nabáda softvérových inžinierov k takému správaniu, ktoré vytvorí zo softvérového inžinierstva užitočnú a rešpektovanú profesiu [1].

Učiť sa, učiť sa, učiť sa

Väčšia časť posledného – ôsmeho – princípu Etického kódexu sa zaoberá učením sa a zlepšovaním svojich schopností.

Vedomosti sú hlavnou devízou každého inžiniera a jedine ich rozširovaním sa môže udržať na úrovni potrebnej pre vykonávanie tejto rýchlo sa rozvíjajúcej profesie. Nasledujúci príklad dokumentuje, ako môže byť nevzdelanosť, respektíve zúžený pohľad na danú problematiku príčinou smrteľnej nehody.

Príkladom je esej o robotovi–zabijakovi [2], ktorý usmrtil svojho operátora ako dôsledok chyby v ovládacom programe. Esej je zostavená ako skupina novinových článkov, ktoré čitateľovi prezentujú vývoj v kauze tohto priemyselného robota. Zo smrti jeho operátora bol obvinený programátor ovládacieho programu. Aj keď je esej vymyslená, zvolená forma (novinové články) jej dodávajú pocit pravosti a poslúži nám ako príklad viacerých etických problémov, s ktorými aktéri prišli do styku a na riešenie ktorých sa práve Etický kódex softvérového inžinierstva snaží dať návod.

Z príbehu vyplýva, že jednou z viacerých príčin nehody bol nevhodne zvolený vývojový model. Šéfom projektu sa stal človek, ktorý dlhodobo pracoval na podobných projektoch a používal vodopádový model vývoja softvéru, ktorý sa mu osvedčoval. Tento projekt priemyselného robota však obsahoval nové aspekty, ako návrh používateľského rozhrania, ktoré by vyžadovali skôr prototypovací model vývoja. Šéf projektu odmietal vypočuť hlasy zamestnancov a trval na pôvodnom modeli, pričom si nebol schopný uvedomiť zastaranosť a technickú obmedzenosť svojich názorov, považujúc prototypovanie za akúsi „módnu“ záležitosť.

Jeho neznalosť moderných postupov a neochota rozšíriť si vedomosti boli jednou z príčin nedokonalého návrhu ovládacieho panelu robota, ktorý bol tým pádom ťažšie použiteľný, čo sa tragicky prejavilo v dôležitej chvíli, keď mal operátor jeho použitím robota zastaviť.

Kódex ďalej hovorí o zlepšovaní znalosti doménovej oblasti, v ktorej softvérový inžinier pracuje. Nepoužitie tohto bodu a jeho negatívny dopad sa prejavuje v príbehu v osobe obžalovaného programátora softvéru robota, ktorý sa pomýlil pri prevode vzorcov z tlačenej formy v dokumentácii do formy programu. Ak by mal vedomosti v danej oblasti, nebolo by sa to stalo, pretože by rozumel podstate vzorcov a vedel by, že program je nesprávny.

Samozrejme vyvstáva otázka, akým veľkým odborníkom by sa mal v danej oblasti stať inžinier, aby sa predišlo podobným nepríjemnostiam. Svoj čas musí totiž inžinier venovať najmä štúdiu prostriedkov svojej profesie, a tak bude vždy do určitej miery odkázaný na skutočných odborníkov v príslušnej aplikačnej oblasti.

Etické zastrašovanie

Zamestnávateľom sotvérového inžiniera je väčšinou nejaká komerčná organizácia a jej hlavným zámerom je dosiahnutie čo najväčšieho zisku. Tento zámer sa v určitých

prípadoch môže prejaviť v podobe, ktorá je nezlučiteľná so svedomím inžiniera pracujúceho na projekte, ktorého sa to týka.

V príbehu o robotovi vykonáva riaditeľ divízie robotov nátlak na svojich podriadených, aby mu prehliadnutím skutočnosti, že program obsahuje chyby umožnili dodať tento odberateľovi v stanovenom čase, čím porušuje bod 8.08 kódexu. Pre pracovníčku zodpovednú za testovanie to znamenalo, že vyhotovila falošné testovacie výstupy, aby program prešiel firemnou kontrolou, čím porušila minimálne firemné predpisy. Výsledok nasadenia programu už poznáme.

Proti zastrašovaniu od nadriadených sa ťažko bráni, pretože sa nie je vždy o koho oprieť – ochotní spolupracovníci sa ťažko hľadajú. V roku 1975 IEEE vykonala historický pokus o zriadenie právneho precedensu, keď sa vložila do súdneho procesu troch inžinierov proti firme, ktorá ich prepustila z dôvodu poukázania na nebezpečný návrh systému. Aj keď bol prípad zamietnutý, prerušilo sa ticho okolo praktikovania etiky a veľa súdov v ďalších prípadoch bralo tento aspekt do úvahy [3]. Týmto IEEE dala veľmi podstatný signál svetu, že okrem osobnej statočnosti a profesionality inžinierov je tu ešte iná sila, ktorá ich môže podporiť pri etiky sa týkajúcich konfliktoch so zamestnávateľmi, čím sa zvýšilo odhodlanie inžinierov uplatňovať etické zásady v praxi.

Záver

Posledný princíp etického kódexu by mohol byť vlastne jeho prvým a základným princípom, pretože vyslovuje požiadavky (posledné štyri body), ktoré kladú dôraz na základné vnútorné kvality a úroveň softvérového inžiniera. Práve na týchto základoch by potom mali stavať ostatné princípy, ktoré opisujú jeho správanie sa k okoliu.

Ak by totiž zainteresované osoby v príbehu o robotovi vnútorne súhlasili s etickým kódexom, alebo mali aspoň nejaké vnútorné etické zásady, neboli by konali tak, ako konali, a nebolo by prišlo k vedomému vypusteniu chybného programu.

Použitá literatúra

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

[45] EPSTEIN, Richard G.: The 'Killer Robot' Story, The Online Ethics Center For Engineering & Science, 1998, http://onlineethics.org/cases/robot/robot.html

[46] ELDEN, Walter L.: Curtailing Ethical Harassment, The Institue (IEEE), 1996, http://www.institute.ieee.org/INST/feb96/ethics.html

Záver

V predchádzajúcich príbehoch sme načrtli problematiku etického kódexu pre softvérových inžinierov. V jednotlivých kapitolách sme sa zaoberali problematikou vzťahov odborníkov k ich práci, kolektívu, k verejnosti, posudzovali sme vzťahy zamestnanca a zamestnávateľa a mnoho ďalších. Položili sme mnoho otázok a na mnoho otázok sme sa snažili nájsť aj odpovede. Výsledok však smeruje iba k jednému poznatku:

Etické princípy sú aktuálne vo všetkých oblastiach spoločenského života a nevyhneme sa im ani v čisto odborných sférach, akou je nepochybne softvérové inžinierstvo.

Tak, ako dodržiavanie základných princípov morálky oddeľuje ľudí na dobrých a zlých, zavádzanie etických kódexov prináša snahu o „zušľachtenie“ odbornej práce a viditeľné zlepšenie imidžu firmy a jej ľudí, ktorí sa zaviazali princípy etiky dodržiavať. V súčasnej dobe si už žiadna významná firma nemôže dovoliť nezaoberať sa otázkami súvisiacimi s etikou a jej presadzovaním v praxi. Podľa štatistických výskumov až 93 % zahraničných firiem potvrdilo, že majú zavedené etické kódexy. Na Slovensku sa toto číslo pohybuje okolo 20%[2], čo ukazuje, že väčšina našich spoločností zatiaľ dáva jednoznačnú prednosť finančnému profitu pred presadzovaním morálky.

Jedna vec je zavedenie etických kódexov, druhá ich aplikácia v praxi. Žiadneho človeka ešte formálny súhlas s princípmi etiky neurobil lepším. O ich uplatnení môžeme hovoriť až vtedy, keď budeme nielen tvrdiť, že ich dodržiavame, ale až vtedy, keď sa vnútorne presvedčíme o ich správnosti a začneme sa nimi riadiť aj v našej každodennej práci.

---

[2] Podľa Remišová, A.: Etické kódexy. Bratislava: Ekonóm, 1998.

DIEL II.

Odborné články na tému:

Nové trendy v procesoch vývoja softvéru

Úvod

Softvér sa vyvíja už skoro 60 rokov, odkedy IBM ukázala svetu prvý počítač Mark-1 (rok 1944). Samozrejme, že mohutnejší vývoj softvéru zaznamenávame s príchodom osobných počítačov. Možno povedať, že mohutnosť vývoja softvéru je viac-menej priamo úmerná vývoju hardvéru. Z rozvojom hardvéru a jeho masívnou distribúciou sa zväčšilo aj pole zákazníkov a s tým aj záber ich potrieb a požiadaviek na softvér. Rozširovaním Internetu sa tento fakt ešte viac zvýrazňuje.

Tak ako sa vyvíjal a vyvíja samotný softvér, vyvíjajú sa aj procesy vývoja softvéru, teda spôsoby ako softvér vyvíjať efektívne a optimálne vzhľadom na požiadavky prostredia či zákazníka.

Vývoj softvéru na profesionálnej úrovni nikdy nie je absolútne neriadený a živelný. Je jasné, že bez akejkoľvek kontroly a pravidiel by skupina akokoľvek nadaných jednotlivcov nemala šancu vytvoriť dostatočne komplexný, integrálny a kvalitný softvér, pokiaľ by sa nedohodla na postupe jeho vývoja. Pri vývoji profesionálneho softvéru možno teda identifikovať proces, ktorým sa softvér vyvíja.

Proces vývoja softvéru zahŕňa viacero pravidiel, postupov, techník a metód. Popisuje ako sa má softvér vyvíjať. Čo je výhodné pri danom postupe použiť. Stanovuje základné ohraničenia vývoja. Popisuje spôsob riadenia vývoja softvéru. Navrhuje sledovanie postupu vývoja. Prípadne iné aktivity spojené s vývojom softvéru.

Takýchto procesov môže samozrejme existovať viacero. Väčšina procesov vychádza z praxe. Avšak hĺbka ich teoretického rozpracovania, teda stupeň do akej miery podrobnosti popisujú vytváraný softvér, sa môže odlišovať. Dôležité je uvedomiť si fakt, že časom sa menia aj podmienky a prostredie v ktorom sa softvér vyvíja a tak často nestačí, že sa jednotlivé procesy len upresňujú. Nové podmienky prinášajú nové požiadavky ako na softvér, tak aj na jeho vývoj. Preto časom vznikajú aj úplne nové procesy vývoja softvéru. V praxi sa potom nemusia pri vývoji použiť procesy vždy len v čistej forme. Niekedy je výhodné v rozumnej miere použiť popri základnom vybranom procese aj prístupy niektorých iných.

Taktiež nemusí vždy platiť, že najnovší proces je najlepší proces. V niektorom kontexte môže byť výhodnejší jeden, v inom druhý proces.

V našej publikácií uvádzame prehľad aj starších a známejších procesov vývoja softvéru, ako je klasický vodopádový model. V odborných príspevkoch sa však venujeme aj novším, modernejším a aktuálnejším prístupom k vývoju softvéru.

V článkoch Jehlára a Ďumbiera je uvádzaný prehľad klasických starších a známejších procesov vývoja softvéru.

Dvoran a Milička sa vo svojich príspevkoch venujú znovupoužitiu softvéru. U Dvorana ide o „Komponentvér“, teda budovanie nového softvéru s použitím už existujúcich súčiastok. Milička opisuje populárny prístup „Návrhových vzorov“, pri ktorom sa pri návrhu používajú isté vzory, ktoré boli identifikované ako výhodné pre daný problém už v iných projektoch.

Pucher sa venuje tomu, ako odstrániť časté problémy softvérových projektov a ako vo všeobecnosti vytvoriť optimálne podmienky pre vývoj softvéru.

Černák vo svojom príspevku opisuje možnosti ako minimalizovať náklady vývoja softvéru.

Šimo a Bočkay sa zaoberajú procesmi „Extrémne programovanie“ a „Synchronizuj a stabilizuj“, pre ktoré je charakteristické dynamicky sa meniace prostredie a požiadavky od zákazníka. Tieto procesy sa rozvinuli aj vďaka rozvoju Internetu, ktorý takéto prostredie vytvára.

Internet vytvára výhodné prostredie aj pre proces popisovaný Filkornom „Otvorený kód“, anglicky „Open source“, ktorého charakteristickou črtou je distribúcia vývoja softvéru.

Hanúsek sa zaoberá procesom „Certifikačného testovania“, pri ktorom je kladený zvláštny dôraz na spoľahlivosť a bezpečnosť vyvíjaného softvéru (armáda, kozmický výskum, lekárske prístroje atď.) a nestačí len jednoducho overiť zhodu produktu so špecifikáciou.

Softvérové procesy

Peter Jehlár

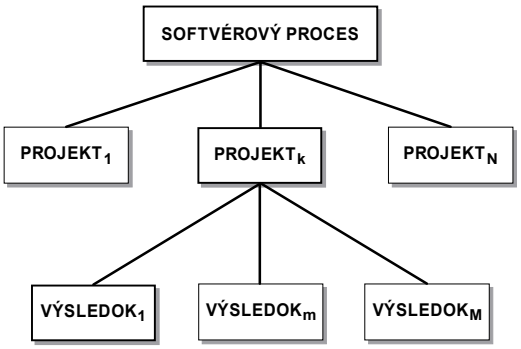
V článku sú definované pojmy softvérový proces, softvérový projekt a ich vzťah. Uvádzame začlenenie procesov vývoja softvéru v rámci softvérových procesov. Definujeme základné vlastnosti procesov vývoja softvéru.

Softvérový proces a projekt

Softvérový proces je proces, ktorý zahŕňa technické aspekty a aspekty manažmentu vývoja softvéru. Softvérový proces určuje abstraktnú množinu činností, ktoré sa majú vykonať pri vývoji softvérového výrobku z pôvodných požiadaviek používateľa.

Softvérový projekt predstavuje vykonanie týchto činností pre špecifické požiadavky používateľa. Konkretizuje činnosti a poradie definované procesom – vytvára projektový plán. Softvérový projekt je časovo ohraničené úsilie, ktoré sa vyvíja s cieľom vytvorenia jedinečného výsledku. Je to množina technických a riadiacich činností, ktoré sa požadujú na zabezpečenie podmienok projektovej dohody. [1]

Výstupom softvérového projektu je výsledok. Vzťah medzi softvérovým procesom, projektom a výsledkom zachytáva obr. 1.



Obr. 1: Softvérový proces a projekt.

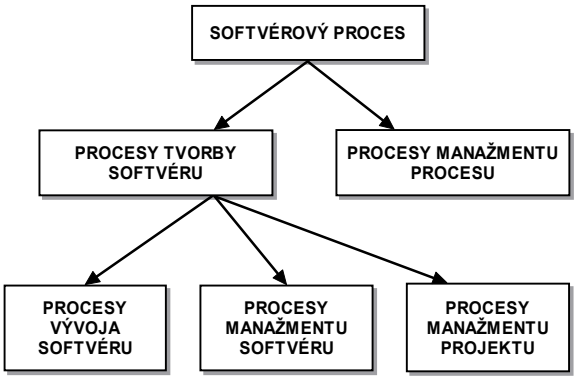
Každý softvérový projekt musí mať definovaný presný dátum začiatku a konca. Musí mať dobre definované ciele a ohraničenia, stanovené zodpovednosti, rozpočet a rozvrh.

Softvérový projekt môže byť samostatný alebo môže byť súčasťou väčšieho projektu. Môže zahŕňať iba časť životného cyklu softvéru, môže trvať niekoľko rokov, môže pozostávať z viacerých podprojektov.

Klasifikácia softvérových procesov

Softvérové procesy možno členiť viacerými spôsobmi. Jeden zo spôsobov zachytáva obr. 2. Procesy možno rozdeliť na procesy tvorby softvéru a procesy manažmentu procesu. [1]

Cieľom procesov manažmentu procesu je zlepšovanie procesov tvorby softvéru. Snažia sa čo najlepšie porozumieť existujúcim procesom. Na základe ich dôkladného poznania, ich upravujú tak, aby sa zlepšili vlastnosti softvéru alebo sa zredukovali náklady a čas na vývoj.



Obr. 2: Členenie softvérových procesov.

Procesy tvorby softvéru možno rozdeliť na procesy vývoja softvéru, procesy manažmentu softvéru a na procesy manažmentu projektu.

Procesy vývoja softvéru predstavujú činnosti, ktoré sú priamo spojené s vývojom softvéru. Patrí sem špecifikácia, návrh, implementácia, validácia, prevádzka a údržba softvéru.

Procesy manažmentu softvéru (manažment softvérových konfigurácií) zabezpečujú riadenie zmien softvérového systému. Identifikujú jednotlivé verzie a konfigurácie počas celého životného cyklu softvéru.

V procesoch manažmentu projektu sa využívajú znalosti, zručnosti, prostriedky a techniky na projektové činnosti. Ich cieľom je dosiahnuť potreby a očakávania projektu.

Charakteristiky softvérového procesu

U softvérových procesov možno identifikovať niekoľko základných charakteristík. Keďže procesy vývoja softvéru sú podmnožinou softvérových procesov, tieto charakteristiky sa týkajú aj ich.

Zrozumiteľnosť udáva ako jasne je definovaný proces a do akej miery je definovaný explicitne. Do akej miery sú výsledky procesu viditeľné charakterizuje viditeľnosť. Prijateľnosť identifikuje, či je definovaný proces prijateľný a použiteľný inžiniermi na tvorbu softvéru. Spoľahlivosť určuje, či sa možno vyhnúť chybám tak, aby sa zabránilo chybám výsledku. Robustnosť procesu vystihuje schopnosť procesu pokračovať aj po výskyte neočakávaných problémov. Udržovateľnosť udáva možnosť zmeny procesu na základe zmeny požiadaviek, ktoré sa vyskytnú v jeho priebehu. Rýchlosť určuje ako rýchlo možno realizovať proces, ktorý na základe špecifikácie vytvorí výrobok – softvér. Ako možno proces podporiť rôznymi softvérovými nástrojmi charakterizuje podporovateľnosť. [2]

Procesy vývoja softvéru

Procesy vývoja softvéru tvoria podmnožinu softvérových procesov. Vyššie sú vybraté niektoré vlastnosti. Už z týchto vlastností je zrejmé, že niektoré si odporujú. Napríklad, ak chceme mať dostatočne viditeľný proces, vytvárame dôkladnejšie dokumentáciu, a tým sa rýchlosť procesu znižuje alebo rastú náklady. Ak chceme docieliť dostatočnú udržovateľnosť, musíme základy systému vytvoriť všeobecnejšie a tým narastá dĺžka procesu vývoja.

Preto existuje veľké množstvo procesov vývoja softvéru a s nimi spojených modelov, ktoré sa podľa potreby zameriavajú na vybrané vlastnosti alebo kombinujú mieru jednotlivých vlastností na základe potrieb.

Použitá literatúra

[47] Bieliková, M.: Softvérové inžinierstvo. Princípy a manažment. Vydavateľstvo STU Bratislava. 2000.

[48] Sommerville, I.: Software Engineering. Addison-Wesley. 5th edition 1996.

Klasické modely

Peter Jehlár

Článok zhŕňa základné charakteristiky týchto modelov procesov vývoja softvéru: vodopádový, V model, inkrementál-ny, evolučný, komponentový, špirálový a model formálnej transformácie. V závere ich stručne porovnáva.

Modely životného cyklu softvéru

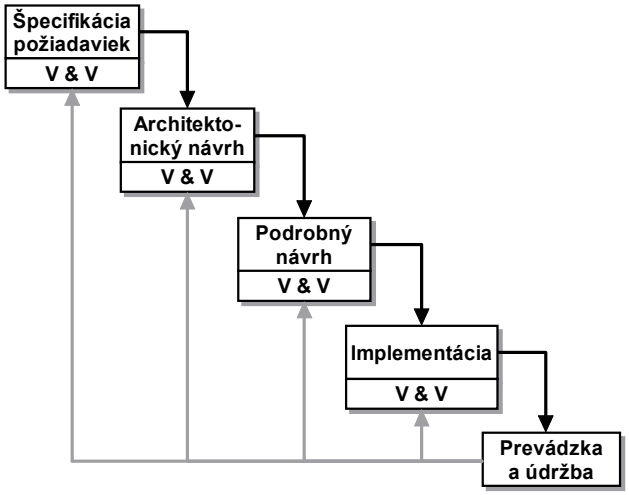
Existuje veľké množstvo modelov životného cyklu softvéru a s nimi spojených procesov vývoja softvéru. Medzi najrozšírenejšie alebo najznámejšie modely patria: vodopádový, V model, inkrementálny, evolučný, komponentový, špirálový a model formálnej transformácie. V ďalších častiach sú stručne opísané jednotlivé, vyššie spomenuté, modely. Pri ich opise vychádzame z diel [1] a [2].

Vodopádový model

Vodopádový model je najstarším modelom. Jeho hlavnou črtou je, že jednotlivé činnosti sa vykonávajú postupne za sebou (obr. 1). Činnosť sa začne vykonávať až po ukončení predchádzajúcej činnosti. Každá činnosť končí verifikáciou a validáciou (V&V). Viditeľnosť procesu je dobrá, pretože každá činnosť končí definovaným výstupom.

Reálne projekty často nedodržujú poradie jednotlivých krokov, kroky sa často vykonávajú paralelne. Jednou z príčin je to, že zákazník nedokáže hneď na samom začiatku sformulovať úplne presne požiadavky na systém. Ak sa postupuje presne podľa modelu, zákazník uvidí systém až na samom konci projektu – môže dôjsť k neskorému odhaleniu nedostatkov, čo môže vážne ohroziť úspech projektu. Model sa nedá prispôsobiť zmenám, ktoré sa vyskytnú.

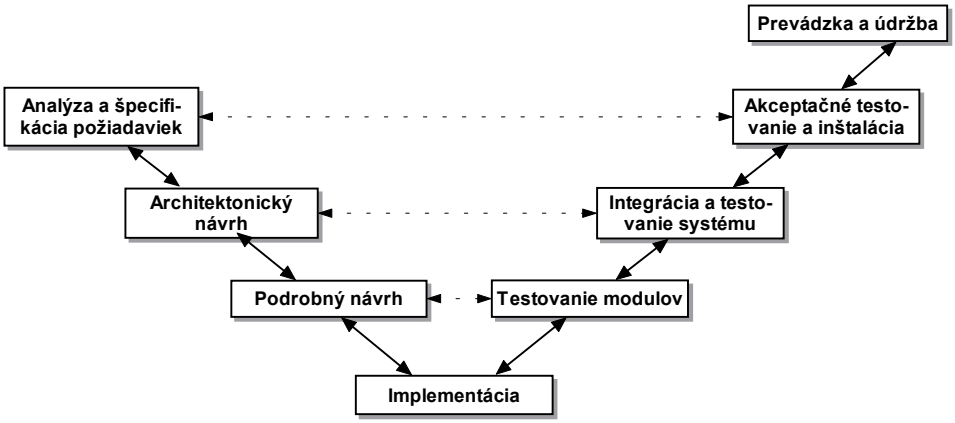
Aj z vyššie uvedeného vidno, že tento model je nevhodný pri vývoji nových druhov aplikácií. Riziko neúplnej špecifikácie a návrhu je vysoké. Na nedostatky sa príde až na konci vývoja. Model je vhodné použiť pri vývoji dobre pochopených aplikácií, kde sa používajú známe postupy riešenia, kde nehrozí riziko, že sa na niečo zabudne.



Obr. 1: Vodopádový model.

V model

V model je obdobou vodopádového modelu (obr. 2). Zachytáva následnosť jednotlivých etáp (aspekt následnosti – plné čiary). Na rozdiel od vodopádového modelu zachytáva aspekt abstrakcie (vzťah testovania s návrhom a analýzou – prerušované čiary).

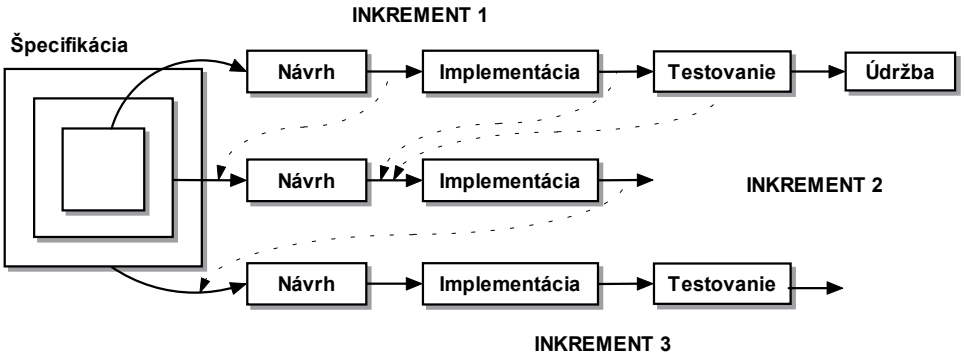


Obr. 2: V model.

Etapy na zostupnej hrane súvisia s dekompozíciou systému na danej úrovni abstrakcie. Etapy na vzostupnej hrane súvisia s integráciou jednotlivých častí systému. V prípade zistenia nedostatku v niektorej z etáp sa možno vrátiť k predchádzajúcej etape alebo k etape na rovnakej úrovni abstrakcie na zostupnej hrane. Viditeľnosť procesu je dobrá, každá činnosť má definovaný výstup, takisto ako u vodopádového modelu.

Inkrementálny model

Vo vodopádovom modeli sa jednotlivé činnosti vykonávajú sekvenčne a až nakoniec je hotový celý softvérový systém. V inkrementálnom modeli (obr. 3) sa na začiatku špecifikujú všetky požiadavky. Zo všetkých požiadaviek sa vyberie podmnožina, na základe ktorej sa vytvorí časť systému – prvý inkrement. K vybratým požiadavkám sa postupne pridávajú ďalšie a ďalšie, čím vznikajú nové inkrementy. Po zahrnutí všetkých požiadaviek je vytvorený celý systém. Jednotlivé inkrementy sa môžu vyvíjať paralelne.



Obr. 3: Inkrementálny model.

Tento model umožňuje znížiť nárazovú zaťaž jednotlivých špecialistov (napr. návrhárov, programátorov). Z hľadiska jedného projektu lepšie využíva zdroje oproti vodopádovému modelu. Kým programátori implementujú návrhármi navrhnutú časť systému – jeden inkrement, návrhári pracujú na rozšírení pôvodného návrhu v nasledujúcom inkremente.

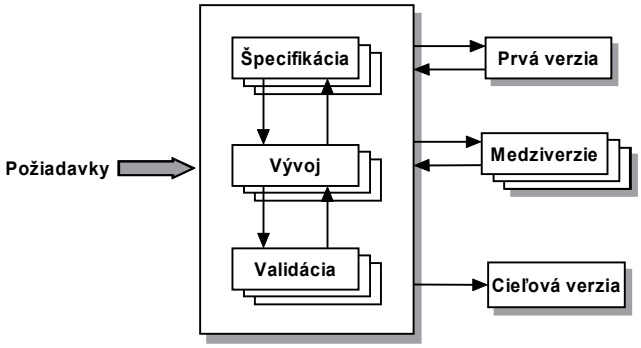
Viditeľnosť procesu je dobrá, pre každý inkrement sa vytvára plán a príslušná dokumentácia. Systém sa vytvára a odovzdáva používateľovi po častiach. Problém môže vzniknúť pri návrhu architektúry, ak je špecifikácia neúplná. Tento model sa používa hlavne pri veľkých systémoch.

Evolučný model

Na rozdiel od inkrementálneho modelu, pri evolučnom modeli (obr. 4) nie sú definované všetky požiadavky na začiatku. Jednotlivé činnosti (návrh, implementácia, testovanie) prebiehajú súbežne. Požiadavky, resp. systém, sa postupne vykryštalizujú na základe medziverzií – prototypov.

Cieľom prototypu na zahodenie je dospieť k porozumeniu požiadaviek na systém. Prototypujú sa nejasné požiadavky. Používa sa ak nevieme dopredu vyjadriť špecifikáciu.

Cieľom prieskumného prototypu je vyvinúť systém v spolupráci so zákazníkom. Prototypujú sa pochopené požiadavky. Je vhodný pri tvorbe používateľských rozhraní.



Obr. 4: Evolučný model.

Viditeľnosť evolučného modelu je slabá. Jednotlivé verzie majú krátku životnosť, preto nemá zmysel robiť podrobnú dokumentáciu. Systémy vytvorené evolučným modelom sú zle alebo málo štruktúrované. Oproti vodopádovému modelu je rýchlejší a lacnejší, hodí sa na systémy s krátkou životnosťou.

Model formálnej transformácie

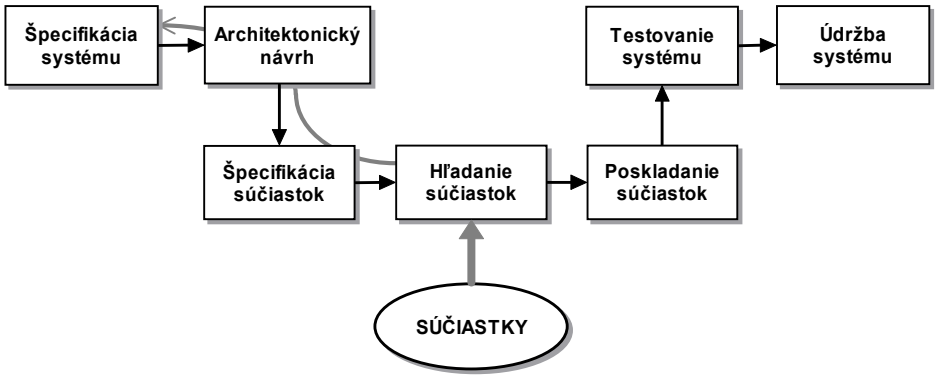
Model formálnej transformácie založený na matematických formalizmoch. Formálna matematická špecifikácia systému sa použitím matematických metód transformuje do programu. Viditeľnosť procesu vývoja je dobrá, v každej etape sa požaduje vytvorenie dokumentu.

V súčasnosti sa tento model používa iba pre malé alebo jednoduché systémy, vtedy keď sú dané formálna špecifikácia a stabilné požiadavky. Tiež je vhodný pri tvorbe vysoko bezpečných aplikácií. Pri použití tohto modelu sa očakáva väčšia zručnosť (z oblasti matematiky) od tvorcov aplikácie.

Komponentový model

Komponentový model (obr. 5) využíva myšlienku znovupoužitia už existujúcich súčiastok systému – komponentov. Na základe špecifikácie sa vytvorí architektonický návrh. Od architektonického návrhu sa odvodia potrebné súčiastky. Tieto súčiastky sa vyhľadajú a poskladajú. Viditeľnosť modelu je stredná. Viditeľnosť je znížená

znovupoužitými komponentmi. Pri použití tohto modelu sa znižujú náklady na vývoj, tiež sa zvyšuje spoľahlivosť systému.

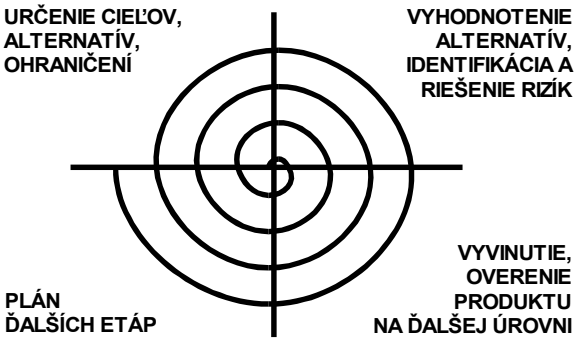


Obr. 5: Komponentový model.

V praxi často špecifikované súčiastky nenájdeme, treba ich vytvoriť alebo modifikovať. Môže sa stať, že je nájdená súčiastka veľmi podobná hľadanej súčiastke, vtedy sa väčšinou upraví špecifikácia. V súčasnosti možno nájsť komponenty, napr., v podobe knižníc a šablón.

Špirálový model

Na rozdiel od predchádzajúcich modelov, má špirálový model (obr. 6) v sebe zahrnuté narábanie s rizikom. Pod rizikom sa myslí niečo, čo môže spôsobovať problémy. Každé kolo špirály predstavuje jednu etapu procesu tvorby softvéru.



Obr. 6: Špirálový model

Kolo špirály sa začína určením cieľov, alternatív a ohraničení pre danú fázu projektu. V nasledujúcom segmente sa vyhodnotia možné alternatívy. Identifikuje sa s nimi

spojené riziko, ktoré sa podľa možnosti čo najviac eliminuje krokmi v ďalšom segmente. V treťom segmente (štvrtý kvadrant) sa vykonávajú činnosti z vyššie spomínaných modelov. Vyššie spomenuté modely možno vložiť do tretieho segmentu, pričom kolo špirály v segmente predstavuje, napr. špecifikáciu, návrh, implementáciu, testovanie, ale aj tvorbu prototypu, hľadanie súčiastok… V poslednom segmente sa na základe výsledkov z predchádzajúcich činností vytvorí plán ďalších etáp.

Viditeľnosť modelu je dobrá, každý segment (kvadrant) špirály v každom kole končí vytvorením dokumentu.

Zhrnutie

Z vyššie uvedeného vidno, že niektoré modely majú veľa spoločného. Niektoré modely sú diametrálne rozdielne. Napríklad, vodopádový je takmer identický s V modelom. V inkrementálnom modeli je vývoj inkrementov zhodný s vodopádovým modelom. V inkrementálnom aj v evolučnom modeli môžeme vytvárať naraz viacero verzií (inkrementov, medziverzií). Pri inkrementálnom modeli na začiatku poznáme všetky požiadavky, pri evolučnom modeli nie. Špirálový model má, na rozdiel od ostatných, v sebe zahrnuté narábanie s rizikom.

Na záver uvádzame v tabuľke 1 vybrané charakteristiky modelov. Stĺpec špecifikácia udáva pri akej špecifikácii, resp. požiadavkách na začiatku procesu je vhodné daný model použiť. Tiež uvádzame typický prípad, kedy je vhodné daný model použiť.

Tab. 1: Vybrané charakteristiky modelov.

Model	Viditeľnosť	Špecifikácia	Vhodnosť použitia
Vodopádový	dobrá	úplná	dobre pochopené aplikácie
V model	dobrá	úplná	dobre pochopené aplikácie
Inkrementálny	dobrá	úplná	rozsiahle aplikácie
Evolučný	slabá	neúplná	dôležité používateľské rozhranie
Komponentový	stredná	úplná	existujúce potrebné súčiastky
Formálnej transf.	dobrá	úplná	formálna špecifikácia
Špirálový	dobrá	neúplná	udalosti spôsobujúce problémy

Použitá literatúra

[1] Pfleeger, S.L.: Software Engineering: Theory and Practice. Prentice Hall. 1998.

[2] Sommerville, I.: Software Engineering. Addison-Wesley. 5th edition 1996.

Evolučný model

Roman Ďumbier

Tento článok predstavuje evolučný model tvorby softvérových systémov, naznačuje výhody jeho použitia a spôsoby nasadenia. Oboznamuje čitateľa aj s možnými problémami pri jeho použití a ich riešením.

Najznámejším modelom procesu tvorby softvérových systémov je vodopádový model. V ňom sú činnosti tvorby softvérového systému zoradené za sebou. Celý proces tvorby je dobre viditeľný, vždy vieme čo práve robíme a ako treba pokračovať. Na konci celého procesu máme hotový program aj s príslušnou dokumentáciou.

Toto je však pravda len v malej časti systémov. Počas tvorby systémov sa totiž podmienky neustále menia a preto takýto proces nie je vždy bez problémov. Problémy vznikajú hlavne s meniacimi sa požiadavkami na systém od zákazníka a okolia systému, ktoré je v neskorších fázach ťažko zahrnúť. Takisto nesprávna funkčnosť programu, prípadne jeho častí je najčastejšie identifikovaná až v posledných fázach jeho vývoja, po jeho úplnej implementácii a integrácii.

Na zamedzeniu týchto problémov vzniklo niekoľko metód, od modifikácií pôvodného vodopádového modelu, až po úplne nové modely softvérových procesov. Jedným z nich je aj popisovaný evolučný model.

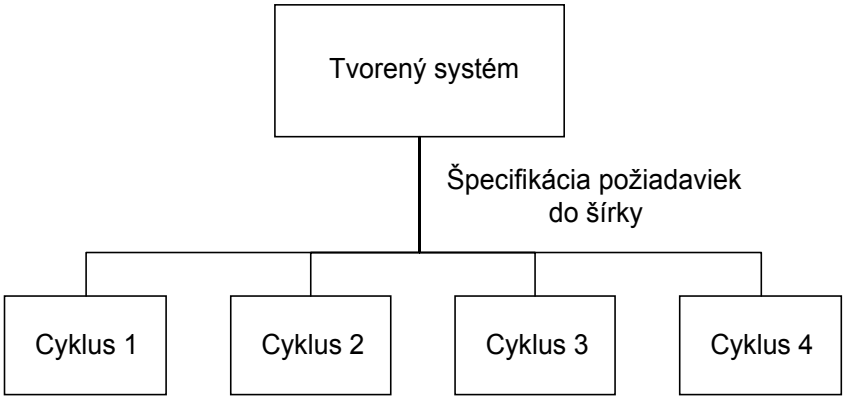
Evolúcia systému

Pri použití evolučného modelu tvorby softvérových systémov sa vychádza z postrehu, že softvérový systém je len zriedka úplne dokončený, vždy je potrebné niečo prerobiť, doplniť alebo zmeniť. Systém sa teda neustále vyvíja, nastáva jeho *evolúcia*.

Pri vodopádovom modeli je výsledné riešenie prezentované až na konci tvorby. Zákazník ho teda môže posúdiť až v tejto konečnej fáze. Ďalší vývoj softvéru sa odohráva až po odovzdaní softvéru. Ak chceme, aby zákazník posúdil náš softvér v skorej fáze, musíme mu ho v nej pripraviť.

Evolučný model vychádza práve z poznatku, že zákazník je schopný posúdiť systém, až keď je úplne hotový, teda až keď je implementovaný. Cieľom je teda vytvárať systém v cykloch tak, aby na konci každého z nich mohol byť systém predvedený zákazníkovi a tým sa zabezpečila dostatočná spätná väzba. Evolučný model sa snaží

eliminovať aj problémy požiadavkami vyvíjajúcimi sa počas tvorby systému tak, že na začiatku procesu sa špecifikujú len hrubé požiadavky na celkový systém (funkčnosť do šírky), presná definícia požiadaviek (funkčnosť do hĺbky) a samotná tvorba systému prebieha samostatne v každom cykle. Postup opisuje obrázok 1.



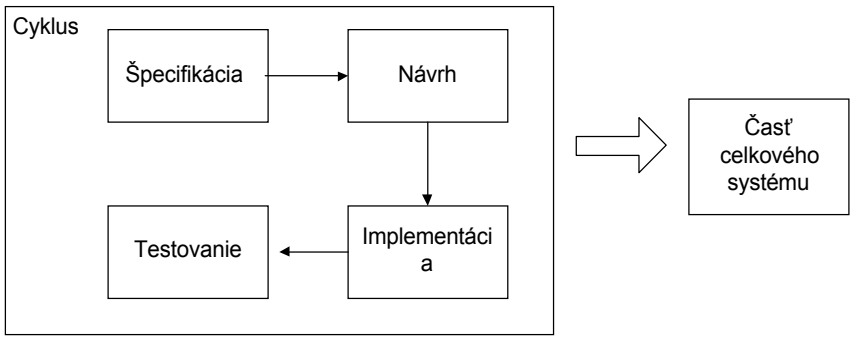
Obr. 1: Špecifikácia v evolučný modeli.

Pri návrhu systému sa na začiatku definujú len pravdepodobné okruhy problémov, ktorými by sa mal softvér zaoberať. Tieto sa nešpecifikujú podrobne, to sa deje až v príslušnom cykle. Často sa totižto stáva, že kvôli zmenám požiadaviek sa vynechá alebo nahradí aj polovica funkčných častí.

Cyklus

Tvorba systému je teda rozdelená na samostatné cykly. Každý cyklus pritom zahŕňa celý proces tvorby príslušnej časti systému, od jeho špecifikácie, cez návrh, implementáciu až po testovanie (obr. 2).

Veľkosť cyklu nie je presne definovaná, odporúča sa, aby jeden cyklus tvoril 5–20 % z celkového času a prostriedkov na celý systém. V praxi to znamená cyklus dĺžky 6–8 týždňov, pri projekte plánovanom na 1 rok. Cyklus by mal mať ucelenú funkčnosť, nemal by obsahovať funkcie bez kontextu (prínos nemusí byť viditeľný zákazníkom vo forme napr. používateľského rozhrania, ten sa môže objaviť až po vytvorení ďalšieho cyklu).



Obr. 2: Cyklus v evolučnom modeli.

Ak by sme vytvorili iba jediný cyklus, dostali by sme sa k známemu vodopádovému modelu. Na druhej strane, ak by bolo cyklov príliš veľa, bolo by ťažké si udržať prehľad o systéme.

Na konci každého cyklu je teda hotový a otestovaný program s prislúchajúcou dokumentáciou. Takýto program sa predvedie zákazníkovi. Systém je pritom v stave, v ktorom by mohol byť celý proces jeho tvorby ukončený a systém odovzdaný zákazníkovi.

Prínosy a nedostatky

Keďže je vývoj systému rozdelený na niekoľko častí, je možné očakávať zlepšenie niektorých vlastností vytvoreného systému. Hlavnými výhodami modelu sú:

- zabezpečenie lepšieho stanovenia požiadaviek, kvôli skráteným časovým horizontom,
- lepšia spätná väzba od zákazníkov, ktorá umožňuje častejšie vytvorenie systému,
- ľahšie napĺňanie stanovených cieľov na krátke obdobie,
- vytvorenie otestovaného produktu po každom vývojovom cykle systému,
- menšie riziko nepodarku, ak sa niečo nepodarí, stačí sa vrátiť o jeden vývojový cyklus späť.

So zlepšením niektorých vlastností však prichádzajú aj nedostatky evolučného modelu, ktoré je potrebné eliminovať, aby bol celý proces tvorby softvéru úspešný. Hlavným nedostatkom je pomerne zlá viditeľnosť procesu riešenia. Keďže nie sú požiadavky špecifikované na začiatku, nie je jasné, či bude vytváraný systém naozaj spĺňať všetky požadované vlastnosti.

Ďalším veľkým nedostatkom je dokumentácia k softvéru. Je pomerne ťažké vytvárať jednotnú dokumentáciu počas celého procesu tvorby systému. Požiadavky na systém sú pritom rozdelené na základné, stanovené na začiatku a podrobné, ktoré vnikli pri tvorbe častí systému a sú teda v dokumentácii vytvorenej počas práce na rôznych miestach.

Jeden cyklus je pomerne jednoduchý a vývojári majú sklon dokumentáciu podceňovať. To sa prejaví až na konci tvorby celého systému.

Použitie

Evolučný model tvorby systémov je vhodné použiť na systémy, kde nie sú dobre známe požiadavky. Jedná sa hlavne o nové typy aplikácií, ku ktorým nie sú dobre známe požiadavky. Jeho použitie je vhodné v podstate pri každom väčšom projekte, kde je teda možné očakávať ťažkosti s určením, prípadne so zmenami požiadaviek.

Jeho nasadenie vyžaduje však určité skúsenosti s tvorbou softvérových systémov a to hlavne od návrhára celého systému. Pri zle navrhnutej kostre je pomerne ťažké vytvoriť aj v úspešných cykloch zodpovedajúci výrobok.

Záver

Evolučný model tvorby softvérových systémov umožňuje pri správnom nasadení dosiahnuť lepšie výsledky v kratšom čase a za nižšiu cenu ako vodopádový model. Pomáha vytvárať systémy lepšie spĺňajúce požiadavky zákazníkov a tým zvyšovať efektivitu práce.

Použitá literatúra

[49] Stuart Woodward, Evolutionary Projekt Management, IEEE Computer, Vol.32, No. 10, 49-57, October 1999.

[50] Bieliková, M.: Softvérové inžinierstvo. Princípy a manažment. Vydavateľstvo STU Bratislava. 2000.

Komponentvér

Robert Dvoran

Súčasný trend globalizácie a celkovej integrácie poskytuje pri tvorbe softvéru rozsiahle možnosti pre znovupoužitie existujúcich komponentov, zároveň však kladie vyššie požiadavky na vytváranie modulárnych a flexibilných systémov, ktoré by tento prístup umožňovali. Tvorba znovupoužiteľných systémov je spojená s vytvorením vhodného modelu procesu vývoja, pretože tradičné prístupy často v nových podmienkach zlyhávajú.

„Hlavným cieľom komponentvéru je vytvorenie vhodne štruktúrovaného systému z nezávisle pochopiteľných a použiteľných častí“ [1]. Tvorba systému z overených a používaných súčiastok nie je v inžinierskych disciplínach žiadnou novinkou. V oblasti softvérového inžinierstva, relatívne mladej disciplíne, však pre úspešné využitie tejto myšlienky zatiaľ neexistuje jednotná a systematická terminológia a metodológia. Využitie komponentvéru v praxi by sa malo opierať o pevné piliere vo forme definície formálneho modelu systému, opisných techník, modelu procesu vývoja a rozsiahlu podporu CASE prostriedkami.

- formálny model systému – Obsahuje definície základných pojmov pre komponentvér, ako sú komponent, rozhranie a komunikácia. Pojmy a ich vzájomné vzťahy sú popísané jasne a jednoznačne formálnymi matematickými technikami.
- opisné techniky – Aby bolo možné formálne definovaný model aplikovať v praxi pri tvorbe reálnych systémov je potrebné poskytnúť grafické a textové opisné techniky, ktorých pochopenie je intuitívne a nevyžaduje rozsiahle matematické znalosti potrebné na prácu s formálnym modelom.
- model procesu – Proces tvorby softvérových systémov by mal byť riadený modelom, ktorý zodpovedá požiadavkám a špecifikám vývoja softvéru so znovupoužitím. Jednou z možností je využitie tzv. procesných vzorov na vytvorenie nového flexibilného modelu procesu vývoja softvéru. Ďalšou alternatívou je modifikácia nejakého existujúceho a používaného modelu, napríklad V-modelu [2].
- podpora nástrojmi – Je potrebné, aby existovali nástroje na podporu tvorby rozsiahlych systémov. Medzi požadované vlastnosti patrí automatické generovanie častí implementácie a dokumentácie, overenie správnosti a

konzistencie popisu systému a verifikácia kritických vlastností systému podľa formálneho modelu systému.

Formálny model systému

Model pre komponentvér sa musí zaoberať nielen statickými vlastnosťami systému, ale je potrebné, aby poskytoval aj nástroje na zápis dynamiky systému a popis správania sa jednotlivých častí [1]. Pri modelovaní systému sa rozlišuje medzi tromi základnými úrovňami popisu, s čím je spojená aj možnosť rôznych pohľadov na systém.

- Inštancie – reprezentujú základné funkčné jednotky systému, ktoré určujú jeho celkové správanie. Tieto sa ďalej rozdeľujú na inštancie komponentov, rozhraní a spojení. Definovaním určitých vzťahov a podmienok pre množiny inštancií sa vytvára priestor pre modelovanie používaných prístupov ku tvorbe komponentvéru.
- Typy – zahŕňajú podmnožinu inštancií komponentov, resp. rozhraní s podobnými vlastnosťami. Každá inštancia je priradená práve jednému typu.
- Popisy – charakterizujú jednotlivé typy, a tým aj zodpovedajúce inštancie. Umožňujú popísať signatúru komponentov alebo rozhraní, očakávanú vstupnú a výstupnú komunikáciu, ako aj spôsob spolupráce rôznych komponentov.

Z pohľadu inštancií sú hlavným stavebným kameňom systému komponenty. Každý komponent vlastní množinu rozhraní, ktoré je možné prepojiť s inými rozhraniami prostredníctvom spojení. Jednotlivé časti systému spolu komunikujú zasielaním správ. Jedinou aktívnou súčasťou systému sú komponenty, ktoré produkujú správy, a cez svoje rozhrania a vzájomné spojenia pomocou nich komunikujú.

Vzťahy medzi jednotlivými druhmi inštancií sú definované základnými funkciami. *Funkcia priradenia* asociuje rozhranie s komponentom, teda hovorí o tom, akému komponentu je priradené konkrétne rozhranie. Je to úplná, ale nie nutne injektívna funkcia, čo znamená, že rozhranie nemôže existovať samostatne, teda mu musí byť priradený práve jeden komponent, ale komponent môže mať priradených aj viac rozhraní.

Funkcia prepojenia definuje vzťah medzi spojeniami a rozhraniami. Model nepovoľuje viacnásobné spojenie rozhraní, teda medzi dvomi rozhraniami môže existovať najviac jedno obojsmerné spojenie, povolené je však spojenie jedného rozhrania s viacerými rôznymi rozhraniami.

Poslednou funkciou, ktorá popisuje štruktúru systému je *funkcia zapuzdrenia*. Definuje hierarchický vzťah rodič – dieťa medzi dvomi komponentmi. Táto funkcia je čiastočná, teda nie každý komponent musí mať nejaký nadradený komponent a acyklická, nie je možné, aby nejaký komponent obsahoval sám seba alebo bol súčasťou niektorého svojho sub-komponentu.

Hlavným cieľom zapuzdrenia komponentu je ukrytie implementačných detailov a vytvorenie štandardného „vzhľadu“ komponentu, bez ohľadu na konkrétnu vnútornú štruktúru. Korektný spôsob zapuzdrenia vyžaduje vylúčenie určitých typov spojení. Komponenty môžu byť spolu spojené iba vtedy, ak jeden z nich je priamym rodičom druhého, majú spoločného priameho rodiča alebo nemajú rodiča vôbec. Zavedenie takéhoto obmedzenia zabezpečuje, že nie je možné vytvoriť spojenie cez „hranice“ komponentu a obchádzať tak definované rozhranie komponentu využívaním služieb priamo vnoreného komponentu.

Model definuje aj spôsob správania jednotlivých inštancií. Spojenia sú pasívne prvky systému, ktoré vytvárajú komunikačný kanál medzi pripojenými rozhraniami, nemá teda zmysel uvažovať o spôsobe ich správania. Komponenty a rozhrania vytvárajú akúsi dvojúrovňovú funkčnú štruktúru správania systému. Komponenty zabezpečujú hlavnú funkcionalitu služieb poskytovaných systémom, kým rozhrania vytvárajú akési komunikačné protokoly na ich vzájomnú komunikáciu.

Hlavnou úlohou rozhraní je prijímať správy z pripojených komponentov alebo spojení a vysielať ich ďalej, napríklad ak príjmu správu z komponentu, ich úlohou je vyslať ju cez všetky pripojené spojenia, a naopak správy prijaté zo všetkých spojení poskytovať priradenému komponentu. Komponent okrem vysielania a prijímania správ prostredníctvom svojich rozhraní môže meniť správanie systému aj zmenou jeho štruktúry, môže vytvárať a rušiť svoje rozhrania a im zodpovedajúce spojenia.

Dôležitá je aj požiadavka na dodržanie kauzality v správaní systému, teda že výstup a správanie komponentu alebo rozhrania je v určitom momente dané len jeho aktuálnym stavom a históriou interakcie s okolím.

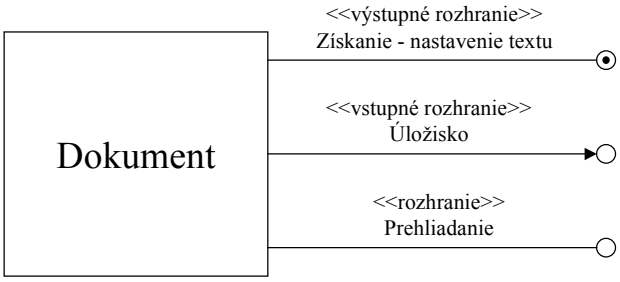
Opisné techniky

Existencia kvalitných techník na zápis vlastností a štruktúry systému je absolútnou nutnosťou pre praktické použitie. Komponentvér je vlastne rozšírením a zovšeobecnením objektovo-orientovaného prístupu k tvorbe systémov, preto aj opisné techniky vychádzajú z postupov používaných v tejto oblasti a sú upravené tak, aby plne vyhovovali požiadavkám komponentvéru [3].

Celkovú štruktúru komponentov zachytáva *diagram typov komponentov* (component type diagram), ktorý vizualizuje jednotlivé typy komponentov spolu s ich rozhraniami. Notácia rozlišuje medzi rozhraniami, ktoré sú určené len na vstup údajov a riadiacich informácií, len na ich výstup a všeobecnými rozhraniami, ktoré podporujú obojsmernú komunikáciu.

Pre ilustráciu uvádzam obrázok 1 ako príklad zápisu komponentu pre jednoduché uchovávanie textu a prácu s ním (dokument). Dokument zapuzdruzdruje textové údaje, aby bolo možné s textom pracovať je potrebné, aby komponent poskytoval výstupné rozhranie pre získavanie a nastavovanie textu. Dokument musí zachovávať perzistenciu údajov, a preto musí poskytovať aj vstupné rozhranie pre pripojenie nejakého úložiska. Okrem toho je potrebné, aby komponent umožňoval definovať určitý pohľad na svoje

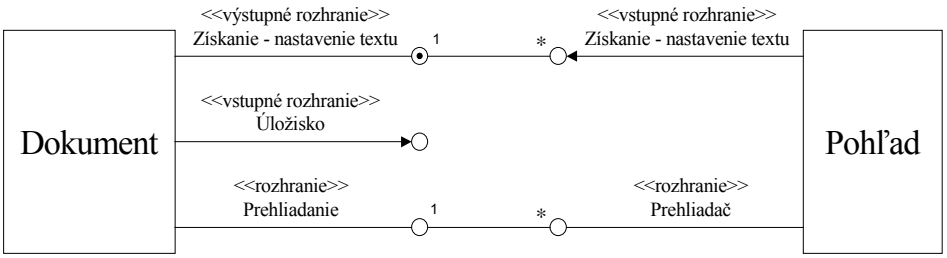
údaje prostredníctvom nejakého zaregistrovaného prehliadača (štruktúra dokument – pohľad).



Obr. 1: Diagram typov komponentov.

Rozšírenou verziou diagramu typov komponentov je *diagram rozhraní komponentov* (interface navigation diagram). Tento špecifikuje rozhrania konkrétneho komponentu a umožňuje zapísať ich viacnásobný výskyt, ako aj pravidlá a následnosť ich použitia pri komunikácii.

Graf inštancií komponentov (component instance graph) popisuje štruktúru prepojení v určitom momente počas behu systému, čo je postačujúce, ak systém dynamicky nemení svoju štruktúru a počet existujúcich komponentov nie je príliš veľký. Ak niektorá z týchto podmienok nie je splnená, je výhodnejšie použiť *graf typov komponentov* (component type graph), ktorý vizualizuje možné spojenia medzi jednotlivými typmi komponentov a kardinalitu týchto spojení, čo je obdoba entitno-relačného diagramu.



Obr. 2: Graf typov komponentov.

Model procesu

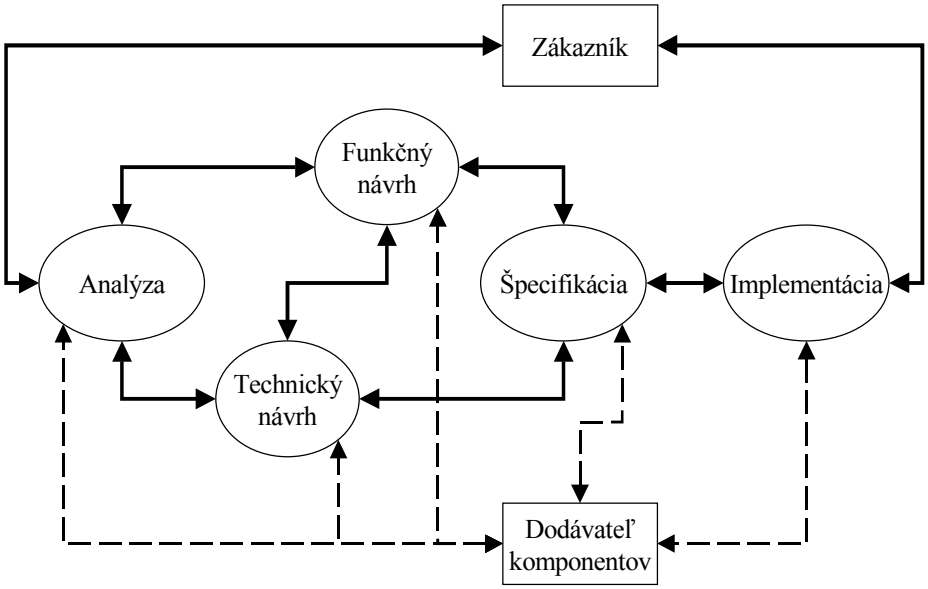
Model procesu podporuje tvorbu softvérových systémov tým, že jasne definuje individuálne vývojové úlohy roly účastníkov procesu a výsledky vývojových úloh, ako aj vzťahy medzi nimi. Tradičné procesy nie sú pre tvorbu komponentvéru použiteľné

z rôznych dôvodov. Najvýznamnejšími sú potreba definovať nové roly a výsledky, ktoré sa v danom modeli nevyskytujú, zvýšenie flexibility modelu kvôli vyššej modularite a prispôsobivosti procesu a nájdenie vhodnej kombinácie prístupu zdola nahor a zhora nadol pri tvorbe systémov. Vývoj systému zdola nahor, ktorý je pre komponentvér prirodzený, nie je v čistej forme príliš vhodný, pretože požiadavky na systém sa berú do úvahy až v relatívne neskorých fázach vývoja.

Využitie procesných vzorov sa v súčasnosti javí ako najsľubnejšia možnosť vytvorenia modelu procesu vývoja komponentvéru. Kostru tvoria tri hlavné časti: štruktúra výsledkov, kritériá konzistencie a procesné vzory [4].

Štruktúra výsledkov

Pod štruktúrou výsledkov rozumieme hierarchickú štruktúru vývojových činností a s nimi spojených výstupov.



Obr. 3: Štruktúra výsledkov.

Obrázok 3 zachytáva štruktúru výsledkov najvyššej úrovne. Oválmi sú reprezentované výsledky hlavných fáz vývoja softvéru. Ich tok medzi fázami je znázornený plnými čiarami. Model nekladie podmienky na vzájomné poradie vykonávania jednotlivých etáp, čím poskytuje požadovanú flexibilitu pri tvorbe komponentvéru. Model definuje ešte jednu úroveň vnorených výsledkov, ich popis by však presiahol rozsah tohoto príspevku. Podrobnejšie ho možno nájsť v [4]. Jednotlivé fázy sú podobné ako pri tradičných procesoch vývoja. Hlavným výstupom fázy *analýza* je špecifikácia

požiadaviek na systém. Jej súčasťou je aj analýza rizík a prínosov vývoja, ako aj prípadný prieskum trhu kvôli zhodnoteniu vhodnosti a návratnosti vývoja systému.

Funkčný návrh definuje celkovú architektúru systému z pohľadu funkcionálnych požiadaviek a komponentov. Zaoberá sa hlavne detailnou špecifikáciou algoritmov, interakcie a zodpovednosti systému a jeho jednotlivých častí v kontexte poskytovaných služieb. *Technický návrh* sa naopak zaoberá definíciou technických komponentov, ich celkovým prepojením a interakciou, ktoré spolu zabezpečujú splnenie nefunkcionálnych požiadaviek na systém. Príkladom by mohlo byť zabezpečenie perzistencie objektov alebo ich distribúcia.

Úlohou fázy *špecifikácia* je spojenie a zosúladenie oboch pohľadov na systém z funkčného a technického návrhu. Súčasťou výsledku by mala byť celková špecifikácia architektúry a komponentov systému, ako aj predbežné priradenie konkrétnych komponentov funkciám systému a ich otestovanie v danom kontexte. Záverečnou, avšak nie nevýznamnou časťou je fáza *implementácia*, ktorá zabezpečí vytvorenie výsledného kódu systému a jeho celkové otestovanie. Na výstupy jednotlivých fáz je možné klásť rôzne prídavné požiadavky, ktoré spolu tvoria kritériá konzistencie.

Procesné vzory

Vzory pozostávajú z troch základných častí. *Kontext* je celková situácia, v ktorej vzniká opakujúci sa *problém*, ktorý je možné riešiť všeobecným a overeným *riešením*. Kontext je definovaný ako súhrn vnútorného a vonkajšieho stavu projektu, na ktorý vplývajú zákazníci, konkurenti a situácia na trhu. Vnútorný stav je charakterizovaný stavom a konzistenciou štruktúry výsledkov. Problém popisuje konkrétnu situáciu, ktorá vzniká v danom vývojovom prostredí a je spôsobená nerovnováhou medzi faktormi určujúcimi kontext, čo vytvára priestor pre nápravnú akciu. Riešenie je aplikácia vzoru na daný problém s využitím častí štruktúry výsledkov a účastníkov procesu v konkrétnych rolách, a poskytuje adekvátnu postupnosť a prioritu vývojových krokov pre danú situáciu.

Novými rolami v kontexte komponentvéru sú rola tvorcu komponentov, ktorého úlohou je rozpoznanie spoločných požiadaviek množstva zákazníkov a vytvorenie vhodného znovupoužiteľného riešenia a rola integrátora komponentov, ktorého úlohou je prispôsobenie znovupoužiteľného komponentu a jeho integrácia do systému.

Procesné vzory je možné rozdeliť do štyroch skupín podľa toho akú veľkú časť vývoja pokrývajú. Pre ilustráciu spomeniem niektoré navrhované vzory, rozsiahlejší a formálny zápis ďalších vzorov je možné nájsť v literatúre [4]. Projektové vzory určujú celkový prístup ku tvorbe systému. V tejto skupine môžeme nájsť zovšeobecnené tradičné prístupy ako vodopádový model s klasickým sekvenčným prístupom zhora nadol, obrátený vodopádový model, ktorý sa aplikuje pri udržiavaní konzistencie výstupov ranných etáp vývoja s celkovými výsledkami, ako aj iteratívny model vývoja systému. K menej tradičným patria architektúrou riadený vzor a vzor „roundtrip“, ktorý je jednorázovou kombináciou vodopádového a obráteného vodopádového vzoru.

Druhou skupinou sú medzivýsledkové vzory, ktoré poskytujú postup pre vytvorenie aspoň dvoch hlavných výsledkov. Sem patria napríklad vzory „ladenie výkonu“, zaoberá sa analýzou výkonnosti komponentov a ich integrácie, a „reverzné inžinierstvo“, ktoré sa analýzou existujúcich zdrojových textov pokúša pochopiť a opätovne využiť čo najviac vhodných vlastností existujúceho návrhu a implementácie. V tejto skupine nájdeme aj vzory skúmajúce dopad a možnosti integrácie nových verzií použitých alebo úplne nových komponentov do vytváraného systému.

Tretia skupina vzorov sa zaoberá niektorým konkrétnym výsledkom hlavnej etapy štruktúry výsledkov. Vzory popisujú vybrané aspekty a prístupy k tvorbe požadovaného výstupu, napríklad vo fáze analýzy sú to vzory „zákazníkom riadená analýza“, pri zákazníckych produktoch, alebo „trhom riadená analýza“, ak sa jedná o generický systém alebo zákazník ešte nemá ujasnené požiadavky.

Posledná štvrtá skupina vzorov sa zaoberá vytváraním čiastkových výsledkov hlavných činností, príkladom môže byť testovanie technikou čiernej skrinky alebo bielej skrinky vo fáze implementácie.

Komponentvér a komponentový model

Klasický komponentový model procesu vývoja, ktorý je v podstate vodopádovým modelom s modifikovanou fázou implementácie, má oproti opísanému modelu niekoľko nevýhod, ktoré vyplývajú z vlastností vodopádového modelu a znižujú jeho použiteľnosť v reálnom prostredí. Hlavnou nevýhodou je nízka úroveň flexibility tohoto procesu. Sekvenčný prístup k vykonávaniu jednotlivých činností a absencia spätnej väzby v procese sa v kontexte vývoja softvéru so znovupoužitím javia ako nie príliš vhodné. Pri nejakej zmene požiadaviek alebo prostredia komponentový model totiž neumožňuje reagovať iným spôsobom ako začatím úplne nového procesu.

Využitie procesných vzorov v modeli pre komponentvér poskytuje požadovanú úroveň flexibility, pretože je možné vybrať si a aplikovať vzor, ktorý je v danej situácii najvhodnejší, v prípade potreby aj vodopádový, a teda postupovať klasickým spôsobom. Definovaná štruktúra výsledkov navyše inherentne zachytáva potrebu iterácií vo vývoji a kombináciu prístupov zhora nadol a zdola nahor, na rozdiel od komponentového modelu, ktorý je v podstate striktnou aplikáciou prístupu zhora nadol.

Na záver by som chcel podotknúť, že hoci vývoj v oblasti komponentvéru napreduje rýchlym tempom, ostáva ešte veľa práce, kým tento prístup ku tvorbe softvéru bude naplno využiteľný v praxi. Okrem definície formálneho modelu a procesu tvorby komponentvéru je potrebné sa sústrediť hlavne na vytvorenie rôznych podporných nástrojov, či už pre tvorbu, distribúciu alebo hľadanie komponentov, ako aj čiastočnú automatizáciu jednotlivých fáz vývoja rozsiahlych softvérových systémov.

Použitá literatúra

[51] Bergner, K., Rausch, A., et al. – A Formal Model for Componentware. In: FBT 99, Proceedings of the 9. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme, June 1999

[52] Ansorge, D., Bergner, K., et al. – Managing Componentware Development - Software Reuse and the V-Modell Process. In: *Lecture Notes in Computer Science 1626, Advanced Information Systems Engineering*, s. 134-148, ISBN 3-540-66157-3, Springer Verlag, June 1999

[53] Bergner, K., Rausch, A., et al. – An Integrated View On Componentware - Concepts, Description Techniques, and Development Process. In: *IASTED 98, Proceedings of IASTED Conference on Software Engineering*, s. 77-82, ACTA Press, October 1998

[54] Bergner, K., Rausch, A., et al. – A Componentware Development Methodology based on Process Patterns. In: *PLOP 98, Proceedings of the 5th Annual Conference on the Pattern Languages of Programs*, August 1998

Návrhové vzory a ich prínos

Tomáš Milička

Článok pojednáva o problematike návrhových vzorov v súvislosti s procesom vývoja softvéru. Najprv vysvetľuje problematiku vzorov vo všeobecnosti. Ďalej opisuje spôsob uchovávania vzorov a prezentuje spôsob ich opisu v najznámejšom katalógu návrhových vzorov. V bodoch opisuje i ich prínos pre návrhára a v závere vymenúva niektoré problémy brániace ich masovému používaniu.

Úvod

"Návrhársky vzor opisuje problém, ktorý so znova a znova opakuje. Opisuje jadro riešenia tohto problému takým spôsobom, že toto riešenie môžeme použiť opätovne miliónkrát bez toho, aby sme to robili dvakrát rovnakým spôsobom."

Ch. Alexander

Christopher Alexander bol architektom a jeho vzory sa stali inšpiráciou pre softvérové inžinierstvo [1]. Myšlienka zozbierať a odovzdávať dobré návrhárske riešenie je zrejmá i v ostatných inžinierskych disciplínach, v ktorých sa ich použitie (resp. znovupoužitie) stalo samozrejmosťou. Vzory v softvérovom inžinierstve pomáhajú priblížiť sa k ostatným inžinierským disciplínam, ktorých základom dobrého výsledku je aplikácia overených inžinierskych riešení.

Čo sú to návrhové vzory ?

Za vzdialených predchodcov vzorov môžeme považovať objekty. Tie priniesli úžasný pokrok v programovaní ako i v znovupoužití softvérových súčiastok. Za logických nasledovníkov objektov v tomto vývoji môžeme považovať objektové návrhové vzory. Vzory sa presadili najprv v objektovo – orientovanej komunite, už naučenej aplikovať znovupoužitie súčiastok (resp. objektov), a neskôr vznikli i v niektorých ostatných oblastiach. Postupom času sa začali vzory aplikovať i pri návrhu, analýze a dnes vidieť vzory v rôznych podobách vo viacerých doménových oblastiach.

Teda aby sme to zhrnuli, návrhárske vzory pomenúvajú a identifikujú opakujúce sa problémy v objektovo - orientovanom návrhu a navrhujú preň riešenie tak, aby sa dalo aplikovať pri tvorbe vlastného softvérového systému. Vzor napomáha pri vytváraní objektovo - orientovaného návrhu, pretože identifikuje triedy, inštancie, ich úlohu, vzájomnú spoluprácu a rozdelenie ich zodpovednosti v spolupráci tak, aby sa príslušný problém vyriešil [2].

Kde sa uchovávajú návrhové vzory ?

Vzory sú v súčasnosti prezentované v rôznych katalógoch (či už v elektronickej alebo papierovej podobe), kde sú opísané pomocou textu a grafických techník. Sú vytvorené rôzne katalógy pre rôzne doménové oblasti. Prezentácia opisu katalógu vytvoreného skupinou známou pod skratkou GoF (Gang of Four: Gamma, Helm, Johnson a Vlissides) je uvedená v nasledujúcej časti Ako opisujeme návrhové vzory. GoF katalóg sa stal najznámejším a najviac používaným katalógom v oblasti návrhových vzorov. Možno ho nazvať bibliou návrhárskych vzorov, preto ho prezentujeme ako typického zástupcu ostatných katalógov.

GoF vzor tu obsahuje informácie, kedy daný vzor použiť, vysvetľuje podstatu problému, opisuje prvky, ktoré treba použiť pri návrhu, ich vzťahy a taktiež dôsledky vzniknuté po aplikácii vzoru. Ďalej obsahuje ukážku kódu, známe použitie a príbuzné vzory.

Ako opisujeme návrhové vzory ?

Vzor v podaní spomínaného kolektívu sa skladá zo štyroch podstatných logických častí [2]:

Meno vzoru:

Pomenovanie zahŕňa v dvoch - troch slovách návrhársky problém, riešenie i dôsledky. Správne pomenovanie vzoru je veľmi dôležité a to najmä:

– z dôvodu uchovávania tohto mena v návrhárskom slovníku
– z dôvodu uľahčenia komunikácie pri návrhu

Problém:

Opisuje, kedy daný vzor použiť a vysvetľuje obsah problému a problém ako taký. Na ilustráciu problému sa používa tzv. motivačný príklad, t.j. konkrétny návrhársky problém ale i podmienky, ktoré musia byť splnené, aby malo použitie vzoru zmysel.

Riešenie:

Opisuje prvky, t.j. triedy a objekty, ktoré použijeme pri návrhu, ich vzťahy atď. Neopisuje konkrétne implementačné alebo návrhárske riešenie, keďže vzor môže byť použitý v rôznych riešeniach. Miesto toho vzor poskytuje riešenie abstrahujúce od detailov a obsahujúce zovšeobecnené elementy (v našom prípade triedy a objekty).

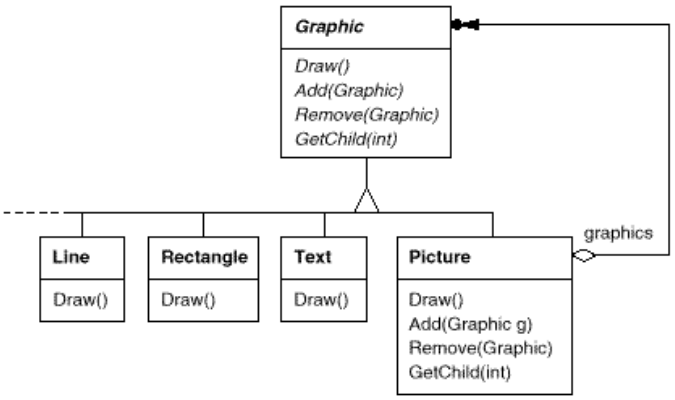
Dôsledky:

Sú nimi výsledky a náklady na riešenie po aplikácii vzoru. Hrajú hlavnú úlohu pri vytváraní alternatív, ako i pri vyhodnocovaní prínosov (ale i tienistých stránok) aplikovaného riešenia.

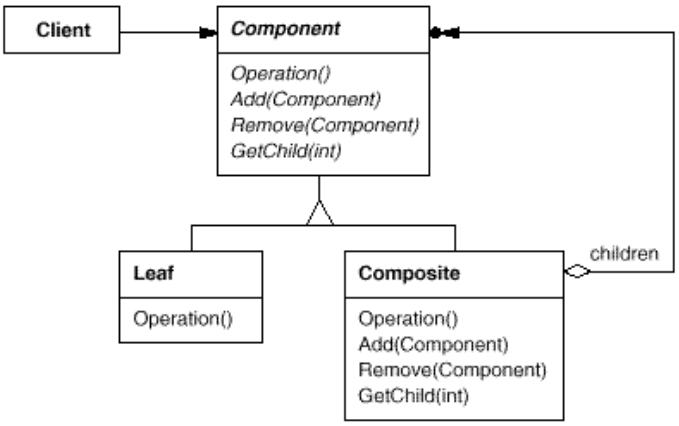
V katalógu sú vzory opísané pomocou určitej šablóny. Je v nej použitý slovný i grafický opis, prípadne ukážka kódu. Okrem spomínaných štyroch častí sa tu nachádzajú i informácie, ktoré robia návrhové vzory ľahšími pre použitie, porovnanie a naučenie.

V GoF použitá šablóna má nasledovné časti:

- *meno vzoru a jeho klasifikácia:* dobre zvolené meno vystihujúce podstatu
- *obsah:* čo vzor robí, jeho obsah, použitie a aké problémy rieši
- *iné pomenovanie:* iné známe pomenovania
- *motivácia:* scenár ilustrujúci návrhársky problém a ako ho štruktúra objektov a tried rieši (ukážka obrázku použitého v časti motivácia vzoru *Composite* – obr. 1)
- *aplikovateľnosť:* kedy vzor aplikovať
- *štruktúra:* grafická reprezentácia tried vzoru v OMT notácii, ukážka štruktúry vzoru *Composite* je na obr. 2
- *participanti:* triedy a objekty participujúce v návrhárskom vzore a ich úloha
- *spolupráca:* ako spolupracujú participanti pri plnení ich úlohy
- *dôsledky:* dôsledky použitia vzoru
- *implementácia:* rady a techniky ako môže byť vzor implementovaný
- *ukážka kódu:* ako by mohol byť vzor implementovaný v C++ a Smalltalk-u
- *známe použitie:* príklad použitia vzoru v reálnom systéme
- *príbuzné vzory:* ktoré vzory sú mu blízke, podstatné rozdiely medzi nimi a s ktorými by mal byť použitý



Obr. 1: Ukážka motivačného príkladu vzoru *Composite*. Znázorňuje použitie vzoru v grafickej aplikácii umožňujúcej používateľovi zgrupovanie menších častí do väčších, ktoré môžu byť opäť zgrupované do ešte väčších častí.



Obr. 2: Ukážka štruktúry vzoru *Composite*. *Component* (*Graphics*) deklaruje rozhranie pre objekty v kompozícii. *Leaf* (*Line*, *Text* a pod.) reprezentuje listové objekty v kompozícii. Listové objekty nemajú deti. *Composit* (*Picture*) definuje správanie pre komponenty, ktoré majú deti a *Client* manipuluje objektmi v kompozícii pomocou rozhrania *Component*.

GoF katalóg je organizovaný podľa dvoch kritérií:

Prvým kritériom je cieľ použitia odzrkadľujúci, čo vzory robia. Vzory môžu byť *vytvárajúce*, *štrukturálne* a *opisujúce správanie*. Vytvárajúce zachytávajú proces tvorby objektov, štrukturálne hovoria o rozložení tried a objektov a opisné charakterizujú spôsob interakcie s okolím.

Druhým kritériom nazývaným miestom aplikácie určuje, či sa vzor aplikuje na *objekt* alebo *triedu*. Vzory triedy pojednávajú o vzťahoch tried a ich podtried a vzory objektov

pojednávajú o vzťahoch objektov. Vzťahy objektov sa môžu dynamicky meniť počas vykonávania programu, na rozdiel od tried a podtried, kde sú vzťahy jasné už pri kompilácii, teda statické.

Aký je ich prínos?

Návrhárske vzory riešia mnoho každodenných problémov, s ktorými sa návrhár stretáva, a to rôznymi spôsobmi. Vymenujeme aspoň niektoré z nich [2]:

- nájdenie patričných objektov
- určenie granularity objektov
- špecifikovanie rozhrania objektu
- špecifikovanie implementácie objektu
- zavedenie mechanizmu znovupoužitia
- súvis "run-time" a "compile-time" štruktúr
- navrhovať s možnosťou neskoršej zmeny
- vytvorenie komunikačného prostriedku v tíme návrhárov aj programátorov

Záver

Problematika návrhových vzorov je pomerne mladá a v súčasnosti prebieha búrlivým rozvojom. Jej masovému rozšírenie a používaniu stále bráni niekoľko problémov. Jedným z nich je napríklad rozptýlenie vzoru v navrhovanom softvérovom systéme po jeho aplikácii. Doposiaľ neexistuje nijaký všeobecne akceptovaný spôsob vizualizácie vzoru po jeho požití, ktorý by riešil problém jeho rozptýlenia. Ďalším problémom je i spätná identifikácia vzoru v už existujúcom softvérovom systéme, ktorý vystupuje do popredia pri re-inžinierstve. Taktiež súčasná forma vzoru nie je vhodná pre bezprostrednú aplikáciu pri riešení návrhárskych problémov.

Ostáva len dúfať, že po vyriešení týchto problémov (a určite i iných so vzormi súvisiacimi) sa stanú návrhové vzory súčasťou každodenného života návrhára ale i programátora.

Použitá literatúra

[55] Alexander, Ch., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language. New York: Oxford University Press, 1977.

[56] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software. New York: Addison Wesley Longman, Inc., 1994.

Modulárny softvérový proces metódou mini-ohodnotenia

Matej Černák

Cieľom tejto metódy je zníženie nákladov ohodnotenia pri kladnej odozve členov projektového tímu. Hoci celkové organizačné hodnotenia môžu byť rozdelené po projektových uzávierkach, metódou mini ohodnotenia je možné primerane načasovať každý projektový tím
Metóda mini-ohodnotenia je založená na modeli vyspelosti procesu CMM, takže za cieľ si dáva aj pochopenie softvérového procesu, vytvorenie konzistentnej bázy pre oceňovanie softvérových procesov a jej výsledkom je vytvorený návrh vývoja softvéru..

Tip 1:

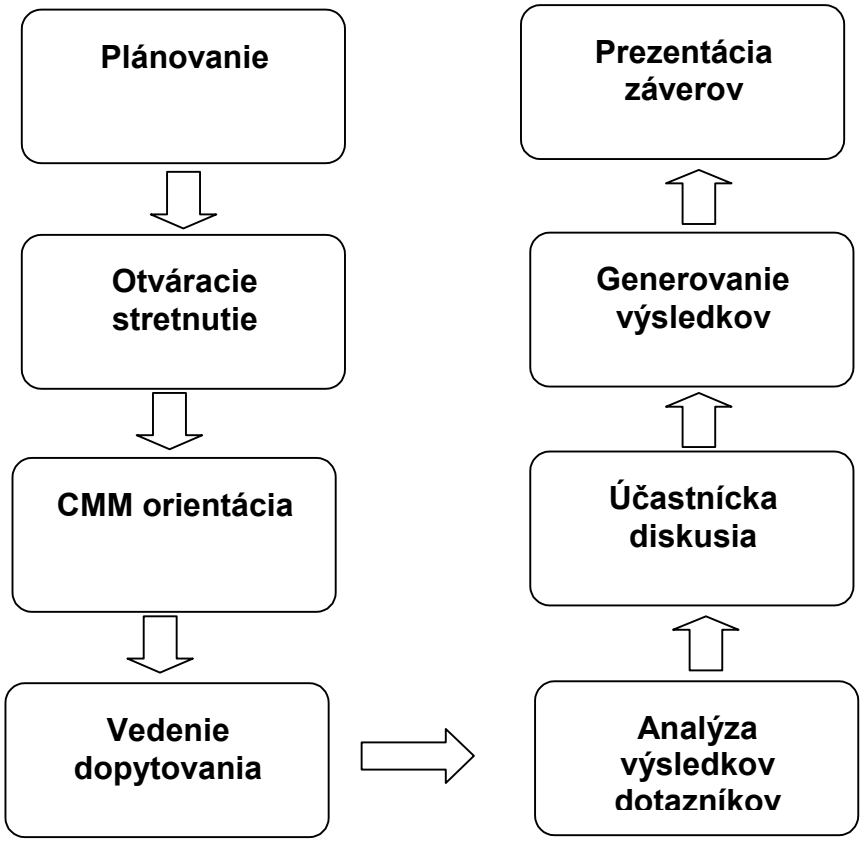
Určiť si ciele softvérového vývoja a zvoliť si techniky odhadu prispôsobené na sledovanie postupu k týmto cieľom.

Opis metódy

Hlavným princípom metód na zhromažďovanie dát využívajúcich metódou mini ohodnotenia (MMA - mini-assessment method) je vyhodnotenie a záver procesných dotazníkov. Následne je možná účastnícka diskusia. Všetci členovia projektového tímu, ktorí sa zúčastňujú na MMA, sa najprv musia orientovať 1 až 4 hodiny v modele vyspelosti procesu (CMM - Capability Maturity Model). Model CMM si dáva za úlohu pochopenie softvérového procesu, vytvorenie konzistentnej bázy pre oceňovanie softvérových procesov a nakoniec poskytnúť návrh vývoja softvéru.

Žiaden z manažérov nadriadených projektovému vedúcemu sa nezúčastňuje tohto zhromažďovania dát. Dohoda o diskrétnosti jasne určuje, že všetky dáta a závery patria projektovému tímu, a že žiadne dáta nie sú pripísané jednotlivcom. MMA navrhuje používať dvoch odhadcov (vedúceho a spätného), hoci jeden odhadca môže urobiť veľa krokov efektívnejšie, čo vedie k zníženiu nákladov.

Obr.1 znázorňuje celkový procesný tok MMA. Tieto základné kroky je možné kombinovať ako aj prispôsobovať za účelom redukovania počtu samostatných udalostí. V každom zo znázornených krokoch je možné uplatniť rôzne postupy.



Obr.1: Schéma procesných tokov modulárneho odhadovania.

Tip 2:

Rozhodnúť, ktorá z metód poskytuje vhodnú rovnováhu medzi cieľmi a subjektívnymi vstupmi, a to ako v procese aktivít vypracovaných projektovým tímom, tak aj v dosiahnutých záveroch tímom.

Flexibilita MMA pramení z viacerých možností, prístupov počas jednotlivých etáp. Tieto možnosti naznačím v každej so spomínaných etapách. Použitím rôznych možností sa ovplyvní počet a aj dĺžku stretnutí. Čas nevyhnutný na mini ohodnotenie sa pohybuje od 2 do 16 kontaktných hodín na účastníka.

Plánovanie

Potom čo sa v projekte rozhodnete použiť metódu mini ohodnotenia, určia sa dvaja odhadcovia. Títo sa stretnú s vedúcim projektového tímu a naplánujú činnosti. Odhadcovia načrtnú informácie o mini ohodnotení a získajú informácie o projekte a tíme. Vedúci tímu povie svoje očakávania od odhadov a odhadcovia vyjadria svoje stanovisko k účastníkom projektu. Odhadcovia pritom zvýraznia, že mini-ohodnotenie je len prvým krokom k zlepšeniu softvérovej spôsoblosti.

Tip 3:

V rôznych bodoch priebehu mini-ohodnotenia, vysvetlite účastníkom ako táto činnosť a jej výsledok vplýva na následné opatrenia, ktoré je potrebné vykonať. Toto pomôže v rýchlosti finálneho ohodnotenia činností procesu softvérového vývoja.

Predloha vysoko prioritných cieľov poslúži ako pomôcka pri plánovaní skupiny vybratých aktivít zameraných na dosiahnutie cieľa

Plánovači vyberú špecifické komponenty. Vedúci súčasne rozhodne, či sa na ohodnotení zúčastní len vybraná vzorka alebo celý tím. Výsledkom tejto etapy je protokol sumarizujúci ciele, účastníkov a udalosti, ktoré boli vybrané pre mini ohodnotenie.

Tip 4:

Rozhodnúť, ktorá podskupina projektového tímu sa zúčastní na poskytnutí dobrých údajov, vlastní výstupy z mini-ohodnotenia a udržuje malé náklady.

Otváracie stretnutie

Zahájenie poskytuje prvé možnosti tímu získať informácie, čo to mini-ohodnotenie je. Ponúkajú sa dve možnosti:

- Samostatná udalosť - ako jedno z naplánovaných stretnutí tímu
- Uvedenie do dopytovanie - krátke informatívne oboznámenie s dopytovaním

Otváracie stretnutie je tiež výborná príležitosť pre vedúceho tímu ako aj jeho nadradených manažérov prezentovať svoju podporu a očakávania od nasledujúcich aktivít.

CMM orientácia

Túto časť možno realizovať až štyrmi spôsobmi:

- 15 minútové prehľadné občerstvenie
- 30 minútové prehľad
- 1 hodinový prehľad o CMM
- 4 hodinový kurz

Odporúča sa použiť dlhšie stretnutie pre nezasvätených účastníkov do metódy MMA. V inom prípade, krátky prehľad je súčasťou inicializácie mini-ohodnocovacích aktivít.

Vedenie dopytovania

Dotazník je dôležitou súčasťou získavania údajov mini-ohodnotením. Dotazník môže vybrať a vytvoriť vedúci projektu ako aj iní účastníci.

Pri vyhodnocovaní dotazníkov sú možné tieto možnosti

- Každý z účastníkov vyplní dotazník
- Vytvorí sa len jedna sada odpovedí

Každá z otázok má možnú odpoveď, ktorá indikuje frekvenciu danej činnosti (vždy, obyčajne, zriedka, nikdy, neviem, ...). Používajte tento postup radšej ako odpovede Ano/Nie. Tiež vyzvite účastníkov k písaniu komentárov k odpovediam.

Druhú možnosť je možné uplatniť len pri obmedzenom počte účastníkov, keďže prísť k spoločnému konsenzu pri väčšom počte je prakticky nemožné. Samostatné odpovede (samozrejme anonymné) poskytujú širšiu škálu odpovedí a názorov, čo umožní lepšie zvoliť optimálne riešenie.

Tip 5:

Neočakávajte od nových účastníkov procesu softvérového vývoja pochopiť CMM (Capability Maturity Model) do takej miery, aby boli schopní sami zhotoviť precízny a správny dotazník. Riaďte priebeh opytovanie v skupine tak, aby ste pomohli všetkým účastníkom spoločne pochopiť otázky.

Analýza výsledkov dotazníkov

Odhadca analyzuje dotazník pomocou špeciálneho nástroja. Výstupmi sú jednotlivé otázky k prislúchajúcim bodom rozdelené v percentuálnom ohodnotení odpovedí. (Vždy=100%, obyčajne=75%, niekedy=50%, ...) Každá z odpovedí má pritom rovnakú váhu. Z každého takéhoto kľúča, množiny otázok, sa potom vytvorí priemer pre výpočet celkovej percentuálnej hodnoty spokojnosti.

Tip 6:

Prevažujúcim cieľom dosiahnutie určitej úrovne CMM, zaznamenajte zmysel dotazníkového hodnotenia a spôsob hodnotenia úrovne. Zamerajte sa na zlepšenie možností, ktoré predložili výsledky dotazníkov ako aj na spísané účastnícke komentáre.

Následne odhadca študuje dotazníkové odozvy a účastnícke komentáre, z ktorých vytvorí zoznam postrehov o každom kľúči projektu. Ak je diskusia naplánovaná, potom tieto postrehy tvoria predbežné závery, ktoré odhadca prezentuje na začiatku diskusie. V prípade, že diskusia nie je plánovaná, odhadca zapracuje postrehy do intenzity procesu a záverečného vyhlásenia.

Účastnícka diskusia

- Táto etapa poskytuje tieto možnosti:
- Žiadna diskusia
- Diskusia na vybrané body
- Diskusia ku hociktorej procesno blízkej otázke

V tejto doplňujúcej aktivite, odhadca vedie diskusiu s členmi projektového tímu za možnej účasti vedúceho. Odporúča sa venovať čas tejto diskusii, ak to projekt umožní. Dodatkovo zistená informácia poskytne kompletný obraz o stave projektu.

Dôležitou súčasťou účastníckej diskusie je potvrdenie významu dotazníkových výsledkov. Ohodnotenia, ktoré sa spoliehajú len na dotazník sú menej dôveryhodné, keďže účastníci mohli chybne interpretovať niektorú otázku alebo poskytli sériu odpovedí, ktorá sa nedá zovšeobecniť. Tieto nepresné odpovede potom redukuje odhadca na základe poznania projektového tímu.

Tip 7:

Hoci výsledky dotazníka a postrehy odhadcov poskytujú dôležitú základňu pre účastnícku diskusiu, počúvajte pozorne "bolestivým bodom" vyjadrených počas diskusie a použite ich ako dodatočné vstupy k záverom.

Generovanie výsledkov

Počas plánovania sa rozhodne, či záverečné stanovisko vytvorí odhadca samostatne alebo závery koncipujú spolu s projektovým tímom. V prvom prípade odhadca upraví postrehy na základe informácii z diskusie, v tom druhom musí odhadca počas diskusie primeť skupinu ku konsenzom. V oboch prípadoch sa doporučuje stanoviť maximálne tri závery ku každému skúmanému kľúču. Každý z týchto záverom identifikuje relatívne procesné slabiny, aktuálne stavy alebo význam týchto slabín a odporúča spôsob ako ich riešiť.

Tip 8:

Počet záverov by mal byť malý, čo zabráni zaplaveniu projektového tímu a umožní zamerať zlepšenia len na zopár vybraných oblastí. Zameranie je kľúčovým slovom procesu softvérového vývoja.

Tip 9:

Vykonávajte autentické kontroly pre každé odporúčanie odporučené zhodnotením, či skutočne pomôžu určiť príslušné závery, a či projektový tím je skutočne schopný implementovať ich, ak sa tak rozhodol.

Prezentácia záverov

Predstavuje posledný krok v mini odhadovaní a prezentuje sumárne závery príslušnému obecenstvu. V prípade, že odhadca vypracoval závery samostatne, prezentuje ich projektovému tímu. Ak závery vypracoval tím, môžu ich prezentovať svojmu manažmentu. Detailnosť prezentovaných záverov je ponechaná na tíme.

Počas prezentácie ukončíme samostatné mini-odhadovanie a opäť zdôrazníme potrebné ďalšie kroky, ktoré musí ešte projekt vykonať. Projekt, ktorý ukončí mini-odhadovanie, avšak nepokračuje ďalšími aktivitami alebo je nemožné zrealizovať naplánované akcie, vytvára dojem neserióznosti manažmentu ohľadom skutočných zmien práce ľudí.

Tip 10:

Ušite si všeobecné materiály na špeciálne prípady, ale vystríhajte sa ich prílišnému priblíženiu. Môžete stratiť ušetrenie nákladov z opätovného použitia, ak použijete prílišne rutinné krajčírstvo.

Vhodným sa javí aj použitie už overených nástrojov a prvkov z iných projektov a vytvorenie akejsi databázy postupov, avšak samotný výber z tejto databázy musí byť uvážený a na základe skúseností.

Praktické skúsenosti

Firma Eastman Kodak Company používala túto metódu na viac ako 24 projektoch. Priemerný počet účastníkov bol 12 členov, pričom rozsah sa pohyboval od 6 do 20. Pružnosť tejto metódy je rozsiahla v odhadcovom ako aj účastníckom snažení, v závislosti od vybraných komponentov. V priemere účastníci strávili 4 hodiny projektových aktivít a odhadca vynaložil celkovo 48 pracovných hodín. Celkové snaženie odhadcu sa znižovalo postupom času, ako sa odhadca zapracúval. Cena takéhoto typického mini-ohodnotenia je približne $6.000, čo je asi 5% až 10% zo Softvérového Procesu Ohodnotenia - Software Process Assessment.

Hodnota modulárnej metódy mini-ohodnotení spočíva v množstve možností, kombinácii modulov, komponentov projektovým vedúcim, čo umožňuje presné prispôsobenie všetkým potrebám a rozpisom.

Členovia projektových tímov sa vo väčšine prípadov stavali pozitívne k tejto metóde: 79% sa vyjadrilo, že čas venovaný MMA bol správny. 52% potvrdilo výpomoc tejto metódy, 37% ešte nevedelo posúdiť a len 11% sa vyjadrilo, že im táto metóda nič nepriniesla.

Použitá literatúra

[57] Karl Wiegers, Doris Sturzenberger: A Modular Software Process Mini-Assessment Method, IEEE Software vol. 16, no. 1

[58] Karl Wieger: 10 Traps to Avoid, Teleconference - March 9 - WWW.TBI.COM/EVENTS/TENTRAPS.HTML

[59] Emanuel R. Baker, Ph.D., Frank J. Koch: SEI Capability Maturity Model, 26.5.2000, WWW.QPMG.COM./SEI.HTM

[60] Paulkm, Curtis, Chrissis, Weber: The Capability Maturity Model For Software, 26.5.2000, WWW.SGI.GU.AU/CMM

[61] M.Daskalantonakis: Achieving Higher SEI Levels, IEEE Software, July 1994, pp. 17-24

Úspech softvérových projektov – kritické faktory

Filip Pucher

Prečo máme stále problémy s vývojom softvérových systémov? Do dnešného dňa sme absolvovali miliardy seminárov o tom, ako správne a načas odovzdať softvérový produkt. Počas rokov sme prešli cez generácie programovacích jazykov, paradigmy vývoja softvérových systémov. Zaviedli sme normy, firemné, projektové, osobné. A stále sme sa neposunuli tam, kde sú stavbári, strojári, chemici. Výskumy, ktoré sme robili a robíme, stále ukazujú, že počet projektov, ktoré majú problémy, sa blíži k 50%. Takže, čo s tým?.

Nie je v tom žiadna mágia, to hlavne. Prečo sa nám nedarí, je dané povahou výrobku – je totiž nezvyčajne komplikovaný. Keďže sa zaoberáme zložitými výrobkami, sami sme zložité, komplikované osobnosti so sklonom komplikovať to, čo robíme. Keď k tomu pridáme, že sa snažíme uspokojiť niečo, čo sa neustále mení – myslím požiadavky užívateľa – tak máme zdroj problémov pomenovaný. Ide o zmes chaotického s nepredvídateľným. A preto sa väčšina projektov v tejto oblasti prediera ťažkosťami s prekročením rozpočtu a odkladaním termínu odovzdania. Aby sme tomu zabránili, treba sa pozrieť na kľúčové faktory, ktoré ovplyvňujú riadenie procesu. Nemyslím odborné, ako manažérske, alebo technické. Mám na mysli ľudské faktory. Takže, päť prirodzených kľúčových faktorov na zvládnutie úspešného softvérového projektu:

1. Vykročme správnou nohou,
2. Udržiavajme zrýchlenie,
3. Sledujme pokrok,
4. Robme správne rozhodnutia,
5. Inštitucionalizujme po-projektovú analýzu.

Ako vidíte, nič nové pod Slnkom. Ide o bežné, prirodzene ľudské záležitosti. Problém je, že nie sú všeobecne zavedené a dodržiavané. Keby boli, nebolo by toľko neúspešných projektov.

1. Vykročme správnou nohou

Nie je ľahké povedať, ktorý z faktorov je najdôležitejší. Ale správne začať je určite dôležité, a to vedie. Analýza projektov ukázala príznaky zlyhania – prinajmenšom sedem z nich sa dá zachytiť pred vznikom návrhu a napísaním jediného riadku kódu. Takže, tu sú:

1. Manažéri nerozumejú požiadavkám používateľa,
2. Zle definovaný rozsah projektu,
3. Slabo riadené zmeny projektu,
4. Vybraná technológia podlieha zmenám,
5. Obchod potrebuje zmenu,
6. Termíny sú stanovené nerealisticky,
7. Používatelia sú nepoučiteľní,
8. Sponzori strácajú záujem,
9. Nedostatok skúsených ľudí,
10. Manažéri ignorujú najlepšie skúsenosti.

Keď máme tieto informáciu, čo môžeme urobiť, aby sme úspešne začali?

1.1 Stanovme si reálne ciele a očakávania

Prvým cieľom, ktorý má projekt dobre odštartovať, je dosiahnuť, aby všetci zainteresovaní boli naladení na rovnakú vlnovú dĺžku. A všetci musia mať reálne očakávania. Aj používatelia. Mali by myslieť na to, že produkt nevyrieši všetky ich problémy, a že prinesie nové. Vývojári musia vedieť, že užívatelia nevedia čo chcú, a že „nedajú pokoj, kým to nedosiahnu“. Často nevedia, koľko peňazí chcú minúť.

1.2 Postavme správny tím

Najprv zistime, či na to máme zdroje. Finančné, predovšetkým. A majmä na pamäti, že ich budeme potrbovať viac, ako očakávame. Postaviť správny tím znamená zohnať vhodných ľudí. Nemali by sme však chcieť, aby všetci boli vynikajúci. Okolo 20% obsadenia tímu najlepším, čo máme k dispozícii, úplne stačí. Príliš veľa hviezd vytvára konflikty ega, no keby sme ich nemali vôbec, tím sa utopí v maličkostiach.

Zostatok tímu by mali obsadiť solídni vývojári s dobrými pracovnými návykmi, schopní spolupracovať. Najkritickejším prvkom pri výbere ľudí je vytvorenie prostredia, v ktorom môžu vyniknúť.

Kedykoľvek to je možné, zapojme zákazníka a používateľa do vývoja. Vytvára to dôveru. Zvyšuje šancu vyvinúť niečo, čo používateľa uspokojí.

1.3 Dajme tímu to, čo potrebuje

Keď máme tím, musíme mu vytvoriť podmienky pre produktívnu prácu. Tiché priestory. Priestory na stretávanie. Formálne aj neformálne. Špičkové výpočtové stredisko. Komfort a „pohodu“. Nešetrime na vybavení.

Vynikajúci tím potrebuje vynikajúce nástroje. Renomované, s podporou. So zaškolením od dodávateľa. Nástroje pre analýzu a návrh, manažment konfigurácií, testovanie, vytvorenie dokumentácie, grafické nástroje. Zakúpenie prvotriednych nástrojov neznamená automaticky investovanie najväčšieho množstva peňazí.

2. Udržiavajme zrýchlenie

Takže máme tím, vynikajúce prostredie, hardvér a nástroje. Nasledujúcou dôležitou úlohou je udržiavať toto zrýchlenie. Jeho prvotné vytvorenie je jednoduchšie, ako jeho udržiavanie. Počas projektu sa často mení. Tieto zmeny sú rýchle, je potrebné na ne okamžite reagovať, aby sme negatívne trendy nahradili pozitívnymi. Tri kľúčové položky zrýchlenia, ktoré je treba sledovať, sú:

1. Fluktuácia pracovníkov – udržiavajme ju nízku,
2. Kvalita – sledujme ju neustále, určujme naše očakávania,
3. Riadenie – riaďme produkt viac ako ľudí.

2.1 Fluktuácia

Stály problém každého zamestnávateľa. Prijme, investuje, zaškolí, a ...stratí. Potom musí zamestnávateľ zohnať náhradu, niekoho, kto je schopný zvládnuť adaptáciu na už rozbehnutý projekt. Kým sa to podarí, plynie čas a vznikajú straty.

2.2 Kvalita

Kvalita sa nedá pridávať spätne. Vo chvíli, keď sa objaví problém s nedostatkom kvality, je väčšinou neskoro na to, aby sa dal riešiť. Urobí sa zmena a vznikne ďalší problém. Spustí sa lavína. Platí, problémy kvality treba poznať vopred.

2.3 Riadenie

Riaďme produkt, nie ľudí. Nakoniec, produkt je to, čo predávame. Takže, nestarajme sa príliš do ľudí. Nechajme ich pracovať tak, ako im je to príjemné. Nechajme ich prísť a odísť vtedy, keď oni chcú. Kritizujme produkt ich práce, nie ich prácu. Ak sa vyskytne skutočný problém s nejakou osobou, riešme ho rázne a rýchle. Nechajme problémového človeka rýchlo odísť a vysvetlime to ostatným.

3. Sledujme pokrok

V „klasickom inžinierstve“ je to jednoduché. Vidím, čo robím. Vidím – sledujem pokrok. Kde nie je fyzická podoba produktu, treba nájsť iný spôsob ukázania progresu. Existuje veľa metodológií na sledovanie postupu „imaginárnych“ prác. Vezmime jednu, vhodnú pre náš projekt a používajme ju.

4. Robme správne rozhodnutia

Správne rozhodnutia často vedú tím od neúspechu k úspechu. A naopak. Nemalo by byť ťažké rozlíšiť správne rozhodnutia od nesprávnych. Napísať niečo napísané odznova za každú cenu patrí medzi tie druhé. Pozorne sledujme technológie, aby sme si nevybrali tie, ktoré už zajtra zaniknú. Snažme sa posunúť naše požiadavky do overenej oblasti, nesnažme sa chcieť niečo, čo tu predtým nebolo. Stavme na overené, odskúšané.

5. Po-projektová analýza

Nie každý je schopný poučiť sa na svojich chybách. Ak sa to nenaučíme, o chvíľu budeme tie isté chyby robiť znova. Určite. Takže, čo s po-projektovou analýzou? Najprv sa pozrime na to, ako boli stanovené a dodržané časové plány. Zistíme, ako vyvíjame produkt, akou schémou sa k nemu blížime. Vytvorme odporúčania a zaveďme ich do praxe. Možno už neurobíme rovnaké chyby.

Päť faktorov, ktoré sa významne podieľajú na vývoji softvérových systémov. Nie je to vyčerpávajúci zoznam – existuje veľa iných faktorov, ktoré ovplyvňujú úspešný manažment. Ale ak zvládneme týchto päť, rýchlo sa zbavíme záporov, ktoré bránia ukončeniu projektu na čas, bez prekročenia rozpočtu. A dáme zákazníkovi to, čo chce.

Použitá literatúra

[1] Reel, J.: Critical Success Factors In Software Projects. IEEE Software, May/June 1999, pp.18-23.

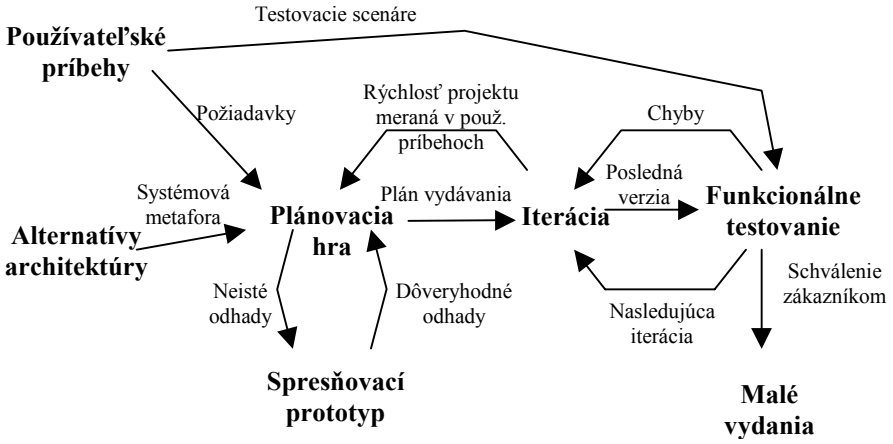
Extrémne programovanie

Branislav Šimo

Tento článok predstavuje čitateľovi spôsob vývoja softvéru nazvaný Extrémne programovanie. Opisuje prostredie, v ktorom ho je vhodné aplikovať, a niektoré základné pravidlá a techniky, ktoré používa.

Úvod

Extrémne programovanie (z anglického *Extreme Programming – XP*) je premyslený a odborný prístup k vývoju softvéru. Kladie dôraz na spokojnosť zákazníkov, umožňuje vývojárom reagovať na zmenu zákazníckych požiadaviek aj v neskorých fázach životného cyklu. Zdôrazňuje tiež tímovú prácu, kde sú všetci – manažéri, zákazníci a vývojári – členmi tímu, ktorý sa snaží dodať kvalitný softvér[3].



Obr.1: Základná schéma softvérového procesu.

---

[3] Pri tvorbe tohto článku som intenzívne používal zdroj [1], a preto ho v texte explicitne neuvádzam. Použitie iných zdrojov je uvedené vždy za príslušnou časťou.

Jeho najcharakteristickejšími črtami sú (z pohľadu manažmentu softvérového procesu) extrémne rýchle iterácie vývojového cyklu softvéru – od definície požiadaviek až po funkcionálne testovanie – a extrémne nízke množstvo vytváranej dokumentácie.

Dôraz sa kladie na komunikáciu, jednoduchosť, spätnú väzbu a odhodlanie. Programátori komunikujú so zákazníkmi a spolupracovníkmi, udržujú návrh jednoduchý a čistý, získavajú spätnú väzbu začatím testovania už v prvý deň, dodávajú systém zákazníkovi čo najskôr a implementujú navrhnuté zmeny.

XP je takzvanou ľahkou (*light*) metodológiou, t.j. metodológiou s málo pravidlami a miernym počtom praktík, ktorými je ľahké sa riadiť. Je to prostredie, ktoré nechá programátorovi priestor pre kreativitu a produktivitu, a pritom je vývoj organizovaný a sústredený.

Poslednou vecou je produktivita. XP projekty jednohlasne prehlasujú vyššiu programátorskú produktivitu v porovnaní s inými projektami v tom istom prostredí. Toto však nebolo cieľom Extrémneho programovania. Skutočným cieľom je dodanie potrebného softvéru v potrebnom čase.

Kedy ho použiť

Extrémne programovanie bolo vytvorené ako odpoveď na problémové oblasti s neustále sa meniacimi požiadavkami. Napríklad ak zákazníci sami nemajú jasnú predstavu o tom, čo by mal systém robiť, prípadne sa vyvíja softvér, u ktorého sa očakáva zmena funkcionality každých pár mesiacov. V týchto prípadoch XP uspeje, zatiaľ čo iné metodológie nie.

XP sa takisto zaoberá problémami rizika. Keď je potrebné vyvinúť nový projekt do určitého dátumu, ak je projekt novinkou pre softvérový tím alebo je to dokonca niečo nové v celom softvérovom priemysle, riziko je veľké. Praktiky XP sú pripravené tak, aby toto riziko minimalizovali a zvýšili pravdepodobnosť úspechu.

Predstavuje však aj prínos do oblastí, v ktorých sa používajú klasické prístupy k vývoju softvéru, akým je napríklad vodopádový model. Na začiatku sú fázy špecifikácie požiadaviek a návrhu, ktorých výsledkom je množstvo dokumentácie. Na tieto dve fázy sa spotrebovalo množstvo času (primerané rozsahu systému), počas ktorého sa vlastný program vôbec nevytváral. Mohlo by sa teda stať, že v čase odovzdávania bude existovať obrovská dokumentácia a len čiastočná implementácia. Pri veľkých projektoch sa tiež môže stať, že kým sa tím dostane k implementácii, požiadavky sa už stihli zmeniť. Riešením je osvojiť si inkrementálny proces získavania požiadaviek a proces inkrementálneho vývoja. Tento prístup umožní podstatne skorší začiatok vlastnej implementácie, čo umožní dodať 30% až 50% funkcionality ešte pred skončením fázy analýzy [2].

Čo sa týka vhodnosti XP pre rôzne veľkosti a typy projektov, v článku [3] sú uvedené rôzne reálne projekty, ktorých veľkosť sa počítala na roky, a ktoré použitím tohto procesu vývoja dosiahli kvalitnejší resp. vôbec nejaký výsledok v porovnaní s inými

procesmi. Iné projekty zas využili len niektoré techniky XP, pričom najdramatickejší vplyv na kvalitu výsledného kódu malo automatické testovanie modulov.

Vytvorenie veľkého informačného systému bude určite vyžadovať počiatočné vytvorenie modelu údajov, nad ktorým bude systém pracovať, čo môže zväčšiť fázu analýzy a špecifikácie, a tým akoby zmenšiť efektivitu XP. Nie je však nutné dopredu špecifikovať všetky funkcie nad týmito údajmi vykonávané, aj keď pre vytvorenie vhodného a efektívneho dátového treba mať o týchto funkciách predstavu. Výhodne sa tu dá použiť základná vlastnosť objektového programovania – zapúzdrenie – takže prípadná zmena dátového modelu sa nemusí katastrofálne prejaviť. Zmenia sa len implementácie niektorých funkcií, nezmenia sa rozhrania, a tak môže zvyšok programu bez väčšej zmeny fungovať ďalej.

Čo vyžaduje

Extrémne programovanie je vhodné pre malé skupiny programátorov, medzi 2 a 10. Rozhodne nie je použiteľné v rozsiahlych skupinách.

Vyžaduje sa širší vývojový tím, čo sa týka zamerania členov, pretože musí taktiež zahŕňať manažérov a zákazníkov pracujúcich bok po boku. Pýtanie sa, zjednávanie rozsahu a termínov a vytváranie funkcionálnych testov vyžaduje viac než len vývojárov.

Ďalšou požiadavkou je testovateľnosť. Musíte byť schopní vytvoriť automatizované testovanie modulov a akceptačné testy. Niektoré oblasti môžu byť touto požiadavkou diskvalifikované, ale prekvapivo veľa ich nie je. V niektorých oblastiach je potrebné uplatniť trocha testovacej vynaliezavosti.

Pravidlá a techniky

Začína sa technikami plánovania, ktorých základ tvoria používateľské príbehy (*user stories*). Tieto môžu byť vytlačené alebo napísané na kartičkách. Rozsah a plán projektu sa vytvorí jednoducho a efektívne prácou s týmito kartičkami.

Nasleduje návrh systémovej architektúry. Hľadajú sa alternatívy architektúry (*architectural spike*) alebo prototypy na vytvorenie jednoduchého celkového návrhu nazývaného systémová metafora. CRC kartičky (*CRC = Class, Responsibilities, Collaboration*) nabádajú členov tímu, aby porozumeli a prispeli k návrhu systému. Špecifickou črtou XP je spoliehanie sa na „prehodnocovanie“ (*refactoring*), ako spôsobu na odkrytie najefektívnejšej systémovej architektúry.

Kvalita kódu je v XP veľmi dôležitá. Techniky zvyšujúce kvalitu kódu sú programovanie v pároch (*pair programming*) a nakódovanie testu modulu pred implementovaním vlastného modulu.

Testovaniu sa prejavuje najvyššia úcta. Dobrý test modulu (*unit test*) a funkcionálny test (*functional test*) sú charakteristickou známkou XP projektu. Projekty v XP

zaujímajú ten postoj, že vývojári sú zodpovední za dôkaz, že ich kód je bezchybný a nenecháva na zákazníkovi, aby dokázal, že kód je chybný.

Zásadou je nepridávať do systému funkcionalitu, o ktorej si myslíte, že bude neskôr využitá. Využije sa len asi 10% pridaných funkcií, takže by ste premárnili 90% času. Extra funkcionalita len spomaľuje a plytvá prostriedkami. Koncentrovať sa treba na to, čo je na programe dnes.

1.1 Používateľské príbehy

Používajú sa na vytvorenie odhadov pre Plánovanie hry (*Planning Game*). Používajú sa namiesto rozsiahlych dokumentov požiadaviek. Píšu ich používatelia ako veci, ktoré chcú, aby ich systém robil. Sú podobné používateľským scenárom, ale nie sú obmedzené na popis používateľského rozhrania. Predstavujú asi tri vety textu napísaného používateľom bez technických výrazov.

Používateľskými príbehmi je riadené funkcionálne testovanie, pretože sa musia vytvoriť jeden alebo viac automatizovaných funkcionálnych testov na overenie toho, že používateľský príbeh bol korektne implementovaný.

Najväčší rozdiel medzi používateľskými príbehmi a špecifikáciou požiadaviek je v úrovni detailov. Používateľské príbehy by mali poskytovať len toľko detailov, aby bolo možné vytvoriť odhad dĺžky implementácie s rozumne nízkym rizikom. Keď príde na vlastnú implementáciu, vývojár zájde za zákazníkom a získa detailný popis požiadaviek.

Odhadom implementačného času sa rozumie „ideálny čas vývoja“ predstavujúci čas, ktorý by trvalo implementovanie daného príbehu, ak by sa nevyskytli žiadne iné úlohy a rozptýlenia. Implementačný čas by mal byť asi 1–3 týždne. Ak je to viac, je potrebné rozbiť príbeh na menšie. Ak je to menej, je to príliš detailné, skúste skombinovať niektoré príbehy. Asi 80 používateľských príbehov je perfektné číslo na vytvorenie plánu vydávania (*release plan*), ktorý špecifikuje, ktoré príbehy budú implementované v ktorom vydaní (*release*) a dátum každého vydania.

1.2 Alternatívy architektúry (prototypy)

Prototypové riešenie (*spike solution*) slúži na zistenie odpovede na ťažký technický alebo návrhový problém. Postaví sa systém, ktorý adresuje len skúmaný problém a ignoruje ostatné aspekty. Väčšinu prototypov si nie je dobré ponechať, preto ho radšej zahoďte. Účelom je redukcia rizika technického problému a zvýšenie spoľahlivosti odhadu používateľského príbehu.

1.3 Systémová metafora

Vytvorenie systémovej metafory slúži na udržanie tímu na určitej úrovni konzistentným pomenovávaním tried a metód. Mená objektov sú veľmi dôležité pre porozumenie celkového návrhu systému a znovupoužitie kódu. Vždy býva užitočné byť schopným správne uhádnuť, ako by sa volala nejaká vec keby existovala; dokáže to ušetriť množstvo času.

1.4 CRC kartičky (*Class, Responsibilities, Collaboration*)

Slúžia na tímový návrh systému. Ich najväčším prínosom je, že umožňujú ľuďom odpútať sa od procedurálneho spôsobu myslenia a plne oceniť objektovú technológiu. Kartičky umožňujú, aby sa do návrhu zapojil celý tím.

Jedna kartička reprezentuje inštanciu objektu. Trieda objektu sa môže napísať na vrch kartičky, úlohy vľavo a spolupracujúce triedy vpravo vedľa každej úlohy. Hovoríme „môže sa napísať“, lebo keď je sedenie v plnom prúde, účastníci väčšinou potrebujú len pár kartičiek s menom triedy a takmer žiadne celkom vyplnené kartičky.

CRC sedenie pokračuje tým, že niekto simuluje systém rozprávaním, ktoré objekty posielajú správy ktorým. Prechádzaním celým procesom sa odhaľujú nedostatky a problémy. Alternatívy návrhu sa môžu rýchlo preskúmať simulovaním predloženého návrhu.

Častou kritikou CRC kartičiek je nedostatočne spísaný návrh. Zvyčajne nie je potrebný, pretože kartičky robia návrh zrejmým. Ak je však potreba trvalejších záznamov, môže sa jedna kartička pre každú triedu plne popísať a uchovať ako dokumentácia.

1.5 Prehodnocovanie

Programátori sa pridržajú svojho starého kódu aj dlho potom, čo sa stal nepraktickým a neudržiavateľným a boja sa ho modifikovať. Otázka znie, či je to efektívne. Extrémne programovanie tvrdí, že nie. Keď sa odstraňuje redundancia, eliminuje nepoužitá funkcionalita a omladzuje zastaralý návrh, prehodnocujeme (*refactoring*). Prehodnocovanie počas celého životného cyklu šetrí čas a zvyšuje kvalitu.

Prehodnocovať treba bez milosti, aby sa predišlo neporiadku a zložitosti. Udržujte svoj kód čistý a stručný, aby sa ľahko modifikoval a udržiaval. Uistite sa, že všetko je vyjadrené len raz. Nakoniec zaberie menej času vyprodukovanie systému, o ktorý je dobre postarané.

Samozrejme je ťažké zanechať už hotový kód a akceptovať nový návrh, ale treba si uvedomiť, že daný kód je zastaralý a vytvoriť nový.

1.6 Programovanie v pároch

Všetok kód, ktorý má byť zahrnutý do projektu vytvárajú dvaja ľudia pracujúci spolu na jednom počítači. Zdá sa to nepravdepodobné, ale dvaja ľudia za jedným počítačom pridajú prinajmenšom toľko novej funkcionality ako dvaja pracujúci oddelene, pričom bude mať tento oveľa vyššiu kvalitu, čo zvýši časovú úsporu neskôr, v ďalších etapách projektu.

1.7 Testovanie modulov

Testy modulov sú jedným zo základných kameňov Extrémneho programovania. Testovanie modulov v XP je však trocha odlišné od bežného testovania. Najprv treba vytvoriť (alebo skopírovať z Internetu) systém pre testovanie modulov, aby bolo možné vytvárať automatizované testovacie balíky. Po druhé, otestovať všetky už existujúce triedy v systéme. Pre ešte neexistujúci kód treba vytvoriť najprv test a až potom vytvárať vlastný kód.

Vytvorenie testov modulov umožňuje vývojárovi premyslieť, čo treba spraviť. Vytvorením testu sa pevne stanovia požiadavky. Testy poskytujú tiež okamžitú spätnú väzbu počas práce a určujú moment dokončenia daného modulu – úspešné vykonanie testu.

Vytvorenie testu a následne kódu sa môže vykonávať v cykloch, t. j. vytvorí sa test pokrývajúci časť problému, ku ktorému sa naprogramuje kód, ktorý ním úspešne prejde. Potom sa vytvorí druhý test, pridá sa nový kód k už vytvorenému, aby prešiel novým testom atď.

Testy modulov umožňujú kolektívne vlastníctvo kódu, t. j. hocikto pracujúci na projekte môže pridať novú funkcionalitu, opravovať chyby alebo meniť kód bez toho, aby to podliehalo kontrole nejakého správcu resp. autora daného kódu. Túto funkciu preberajú automatizované testy.

1.8 Funkcionálne testovanie

Funkcionálne testy sa vytvárajú z používateľských príbehov, pričom príbeh môže mať podľa potreby jeden alebo viac funkcionálnych testov. Testuje sa metódou čiernej skrinky. Keď produkt prejde všetkými funkcionálnymi testami, je pripravený na uvoľnenie.

Každý funkcionálny test reprezentuje očakávaný výsledok zo systému. Za overenie jeho správnosti je zodpovedný zákazník, ktorý pomáha pri vytváraní testovacích údajov a vytvorení správnych vzorových výstupov.

Tieto testy sú tiež automatické, takže je možné ich spúšťať často. Výsledky sa následne zverejnia v tíme, ktorý si musí naplánovať čas na opravenie prípadných chýb.

Záver

Pri súčasných spôsoboch vývoja softvéru sa vývojári stretávajú s rôznymi problémami, ako sú napríklad často sa meniace požiadavky zákazníka, na ktoré je potrebné reagovať, problémy pri integrácii jednotlivých modulov programu a podobne. Extrémne programovanie bolo navrhnuté ako odpoveď na tieto problémy, založené na pozorovaní toho, čo robí programovanie rýchlym a čo ho robí pomalým.

Softvérovým procesom s podobným cieľom je proces „Synchronizuj a stabilizuj" popisovaný v článku Marcela Bočkaya, s ktorým má XP určité spoločné črty, ale rozdiely sú značné, pretože uvedený proces dedí príliš veľa prístupov k tvorbe softvéru zo starších procesov, akým je napr. vodopádový model. Spoločnými črtami sú viaceré cykly vývoja, vytváranie špecifikácie a jej konkretizovanie počas vývoja produktu, malé vývojové tímy. Cykly sú však asi štyri, pričom v XP ich môžu byť desiatky a sú menšie a rýchlejšie. Tímy sú malé, pričom na projekte ich pracuje viac a každý pracuje na určitej črte programu. V článkoch zaoberajúcich sa XP som sa nestretol s projektom, na ktorom by pracovalo viac tímov, čo samozrejme nemusí znamenať, že niečo také neexistuje.

Čo sa týka použitých techník, najväčším rozdielom je základný kameň XP – automatizované testovanie modulov – ktoré v spomínanom procese neexistuje. Hlavná fáza odstraňovania chýb nastáva pri finálnom zintegrovaní produktu, čo môže podstatne sťažiť hľadanie príčin chýb. Tento proces priamo počíta s tým, že nie všetky chyby sa podarí odstrániť, sústreďuje sa len na tie najhoršie. XP sa snaží všetky chyby eliminovať a má na to príslušné prostriedky. Určite to neznamená, že výsledný produkt bude úplne bez chýb, ale bude ich podstatne menej než v predchádzajúcom prípade.

XP je dôležitou novou metodológiou hlavne pre dva dôvody. Prvým a hlavným je prehodnotenie existujúcich vývojových praktík, ktoré sa štandardne používajú. Druhým je, že je jednou z niekoľkých nových tzv. ľahkých metodológií, ktorých cieľom je znížiť náklady na softvér.

Snáď bude teda ďalším krokom na ceste k snu každého vývojára, v ktorom je tvorba softvéru zaujímavou tvorivou hrou a nie utrpením znásobeným približujúcim sa termínom odovzdania softvérového produktu.

Použitá literatúra

[62] DONOVAN WELLS, J.: Extreme Programming: A gentle introduction. http://www.extremeprogramming.org, 2000 (18. 5. 2000)

[63] JEFFRIES, J. E. et al.: XProgramming. http://xprogramming.com, 1999 (18. 5. 2000)

[64] BECK, K.: Embracing Change with Extreme Programming. Computer, Vol. 32, No. 10, 70-77, Oct. 1999

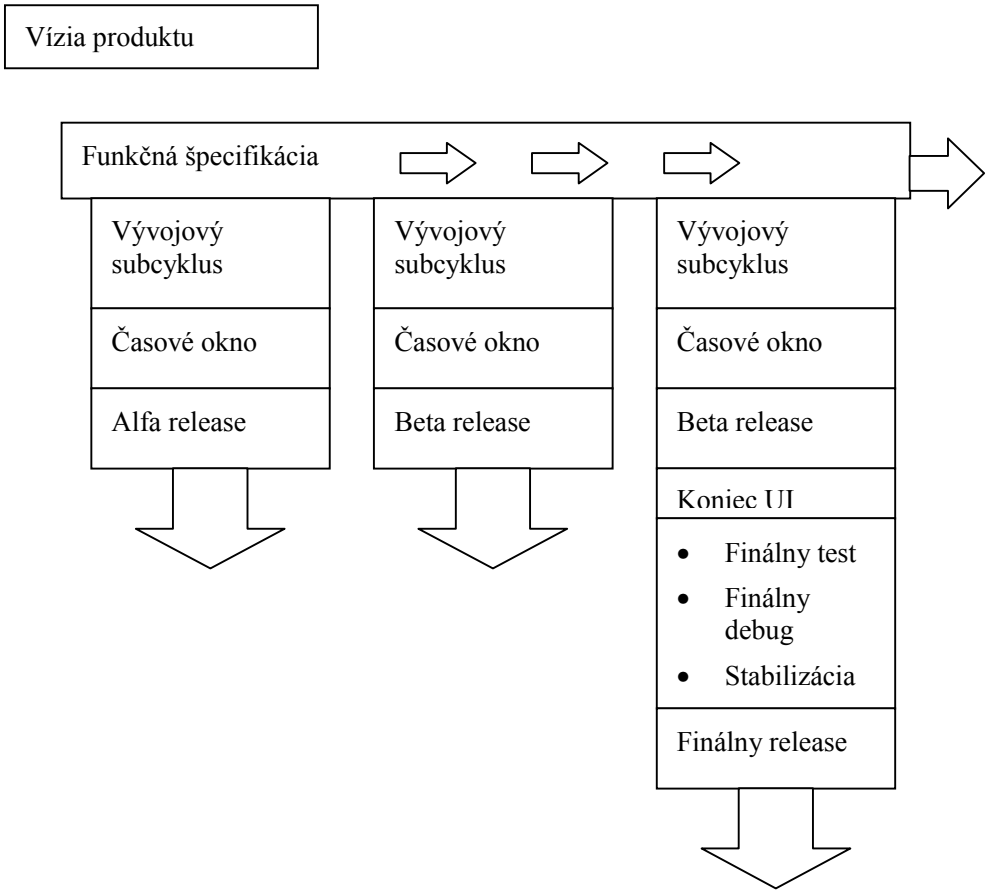
Synchronizuj a stabilizuj

Marcel Bočkay

Žijeme a vyvíjame softvér v dobe Internetu. Vývoj produktov je nevyhnutne iný vo firmách ktoré súťažia v konkurenčnom prostredí nevypočitateľného rýchlo sa meniaceho trhu ako vo firmách so stabilným trhom. Pre dynamické prostredie teda treba vytvoriť proces vývoja softvéru, ktorý bude schopný dostatočne reagovať na jeho požiadavky. Proces Synchronizuj a stabilizuj smeruje k spojeniu a vyváženiu zvyčajnej flexibility a rýchlosti hackera s prístupom profesionálnej inžinierskej disciplíny.

V rokoch 1997-1998 interviovali páni Michael A. Cusumano (MIT Sloan School of Management) a David B. Yoffie (Harvard Business School) [1] viac ako 40 manažérov a inžinierov z firiem Netscape a Microsoft, ktoré sa obe zaoberajú aj vývojom softvéru pre Internet a v prostredí Internetu. Ukázalo sa, že obe firmy, aj keď s istými rozdielmi, používajú pre vývoj softvéru v prostredí Internetu proces, ktorý možno nazvať „Synchronizuj a stabilizuj“.

Viaceré softvérové firmy sa postupne vzdávali zaužívaných sekvenčných procesov vývoja softvéru populárnych najmä pred rokom 1990. Microsoft začal zjemňovať alternatívny proces, ktorý nazývame „Synchronizuj a stabilizuj“ koncom 80-tych a začiatkom 90-tych rokov. Základná idea je dať programátorom dostatočnú nezávislosť pre vývoj návrhu iteratívnym spôsobom a zároveň docieliť, aby členovia tímu frekventovane *synchronizovali* svoju prácu a *stabilizovali* návrhové zmeny, alebo rozšírenia funkcionality.



Obr. 1: Model procesu synchronizuj a stabilizuj
(UI=User Interface, užívateľské rozhranie).

Synchronizuj a stabilizuj

Obrázok 1. znázorňuje jednotlivé časti modelu procesu vývola softvéru nazývaného „Synchronizuj a stabilizuj“. Projekt začína tímovou „Víziou produktu“. Tá naznačuje základné funkcie produktu. Programový manažér potom odvodí hrubú „Funkčnú

špecifikáciu“. Táto sa vyvíja až do konca projektu. Plán projektu obsahuje viaceré stabilizačné body, alebo míľniky. Zvyčajne bývajú tri alebo štyri. Každý stabilizačný bod reprezentuje napredovanie projektu po týždňoch „vývojového subcyklu“, ktorý zahŕňa návrh, implementáciu, testovanie, denné zostavovanie (Daily build), debug, integráciu a stabilizáciu. Tento subcyklus sa opakuje viackrát. Každý míľnik označuje buď alfa, alebo beta release (vydanie). „Časové okno“ (Time buffer), ktoré predchádza každý odovzdávaný systém (release) zabezpečuje flexibilitu tímu vzhľadom na požiadavky dynamického trhu, ako je napríklad pridávanie novej požadovanej funkčnosti do produktu. V záverečnom míľniku sa ukončí vývoj používateľského rozhrania a pridávanie kódu sa taktiež skompletizuje a ukončí. Tím zrealizuje finálne testovanie, odstraňovanie chýb („debug“) a stabilizáciu. Potom sa vydá finálny release. V podstate až keď je projekt kompletný, je funkčná špecifikácia skutočne ukončená. Toto je jednou z výrazných odlišností od tradičného sekvenčného procesu vývoja softvéru (napríklad vodopádový model), kde platí, že ukončená špecifikácia sa nachádza na začiatku vývojového procesu.

Jedným z klasických sekvenčných procesov vývoja softvéru je vodopádový model. Pre lepšie pochopenie procesu synchronizuj a stabilizuj a jeho odlišností od sekvenčných procesov [3] uvádzame ich porovnanie v nasledujúcej tabuľke.

Tab. 1: Porovnanie procesov Synchronizuj a stabilizuj a vodopádového [2].

Synchronizuj a stabilizuj	Vodopádový model
Špecifikácia, vývoj a testovanie sa uskutočňujú (podľa možností) paralelne.	Jednotlivé fázy sa vykonávajú sekvenčne.
Je vytvorený dokument produktovej vízie a špecifikácia sa vyvíja postupne. Špecifikácia je tu výstupom, nie vstupom.	„Kompletný“ dokument špecifikacie a návrhu je vytvorený pred začatím implementačnej fázy.
Prioritné vlastnosti sa budujú v troch, alebo štyroch míľnikoch.	Produkt a jeho časti sa vyvíjajú simultánne.
Frekventovane sa vykonáva synchronizácia („denné zostavenie“ - „daily builds“) pričom v projekte sú 3-4 míľniky.	Na konci projektu je jedna veľká fáza integrácie a testovania.
Dátumy ukončenia sú fixné, ale sú tu viaceré cykly vydávania („release cycles“).	Je tu jeden termín ukončenia a vydania produktu.
Spätná väzba zo zákazníkom je realizovaná počas vývoja.	Spätná väzba zo zákazníkom slúži ako vstup pre budúce projekty.
Je snaha, aby veľké tímy pracovali ako viaceré malé, bez ohľadu na veľkosť	Mnohí jednotlivci pracujú vo veľkých funkcionálnych skupinách.

projektu.

V ďalšej časti sa budeme podrobnejšie venovať podstatným častiam procesu synchronizuj a stabilizuj.

Produktová vízia a funkčná špecifikácia

Typicky na začiatku projektu sa tímy Microsoftu nesnažili spísať kompletnú funkčnú špecifikáciu, ktorá by obmedzila vývojárov na vytváranie fixnej množiny vlastností produktu. Začali vytváraním akejsi vízie, ktorá definovala základné ciele produktu. Tiež pracovali s informáciami od zákazníkov, aby priradili navrhovaným funkciám produktu príslušné priority. Funkčná špecifikácia by mala byť kompletná na konci projektu. Takýto postup bol využitý napríklad pre niektoré verzie Excelu.

Na začiatku začal tím písať funkcie programu, manažéri a vývojári načrtli najdôležitejšie vlastnosti produktu. Tento náčrt by mal byť dostatočne podrobný, aby vedúci projektu mohli určiť rozvrh a zorganizovať tímy zamerané na konkrétne vlastnosti produktu – „tímy vlastností“. Tímy zvyčajne pozostávajú z troch až ôsmych vývojárov (programátorov). Ako vývojári vyvíjajú jednotlivé vlastnosti projektu a vidia tak lepšie čo by mal projekt obsahovať, upresňuje sa ich špecifikácia. Takto sa môže pôvodná špecifikácia na konci projektu líšiť o 30 a viac percent od prvotnej predstavy.

Možnosť neskorších zásahov do špecifikácie považujem osobne za veľmi prínosnú a praktickú vlastnosť procesu. Realizoval som viacero projektov aj klasickým sekvenčným procesom vývoja softvéru. Vo väčšine som sa ale stretol s faktom, že napriek tomu, že špecifikácia mala byť stanovená vopred sa v neskorších fázach procesu (niekedy až pri implementácii) zistilo, že by bolo vhodné niektoré časti špecifikácie vypustiť, pridať, alebo zmeniť v záujme zvýšenia kvality výsledného produktu. A tak sa niekedy stalo, že aj napriek snahe udržať sekvenčnosť jednotlivých fáz a príliš sa nevracať späť, sa nakoniec špecifikácia upravila tak, aby skutočne popisovala výsledný produkt. Z toho vidno, že podrobná špecifikácia má skutočne veľmi často skôr charakter výstupu ako vstupu. Platí to o to viac v prostredí, kde sa môžu požiadavky na produkt a okolie vytváraného produktu meniť v priebehu vývoja.

Míľniky

Celý projekt sa rozdelí do troch, niekedy štyroch častí, ktorých ukončenie nazývame míľnikmi [4]. Tieto reprezentujú kompletizačné a stabilizačné body pre najdôležitejšie množiny vlastností produktu. Každý tím vlastností prejde kompletným vývojovým

subcyklom, zahŕňajúcim integráciu vlastností, testovanie a opravu chýb. Popri tom však tímy, alebo aj jednotliví inžinieri denne synchronizujú svoju prácu výstavbou systému a opravou chýb.

Pred ukončením míľnikovej časti vývojári stabilizujú produkt opravením hlavných chýb a súhlasom nemeniť vybrané čiastkové vlastnosti, alebo meniť ich len veľmi opatrne. Tímy tiež vydávajú interné alfa a externé beta releasy.

Plán projektu tiež obsahuje „časové okná“ (buffer time) zaberajúce 20 až 50 percent celkového vyhradeného času. Tieto okná sa nachádzajú na konci každej časti ukončenej míľnikom, aby členovia tímov mohli reagovať na nepredpokladané ťažkosti, oneskorenia alebo aby mohli pridať neplánované funkcie a vlastnosti produktu.

Časové okná považujem za jednu z charakteristických čŕt procesu „synchronizuj a stabilizuj“, ktorá mu umožňuje byť efektívnym aj v prostredí dynamicky sa meniacich požiadaviek a vyvíjajúcich sa technológií. Programátorom je iste všeobecne známe, že stanovené termíny ukončenia projektu sa zvyknú v poslednej fáze preťahovať najmä kvôli odstraňovaniu chýb a potom ešte niekedy aj odstraňovaniu nových chýb zanesených pri opravách pôvodných chýb. V meniacom sa prostredí ale navyše pribúdajú požiadavky na zmenu, alebo pridávanie nových vlastností softvérového produktu.

Plánovanie

V Microsoft-e produktoví manažéri zvyčajne vytvárali viacúrovňové plány tak, aby sa marketing a inžinieri nesnažili presadiť úplne všetky vlastnosti, ktoré mali pripravené pre čiastkový release. Skúsenejší manažéri a exekutíva dostávali prehľadové správy zhruba každé tri mesiace. Navyše projektový manažéri vypracovávali správu o postupe projektu každý mesiac. Exekutíva mala ešte k dispozícií možnosť porovnávať progres celej firmy vzhľadom k trojročnému strategickému plánu.

Aj Netscape rozvíjal postupne manažment a plánovanie, ako stále viac rástla dôležitosť požiadavky na systematickosť projektov. Dĺžka a intenzita plánovania sa menila s komplexnosťou projektu. Vo všeobecnosti možno povedať, že manažéri a inžinieri Netscape-u prijali filozofiu, že v rýchlo sa meniacom svete Internetu je nemožné špecifikovať všetko, čo by malo byť súčasťou produktu ešte pred začatím písania zdrojového kódu. Taktiež však predpokladali, že isté plánovanie bude nevyhnutné pre tvorbu komplexných produktov.

Denné zostavenie

Zaujímavé je pravidlo denného zostavovania („daily build“), podľa ktorého by sa nový zdrojový kód, alebo akékoľvek zmeny mali aspoň raz denne skompilovať a zostaviť. Microsoft nemá nejaké tvrdo stanovené pravidlo, ktoré by určovalo, ako často majú vývojári synchronizovať svoju prácu, ale každý projekt by sa mal zostaviť

(skompilovať a preložiť – vytvoriť „build“) minimálne raz za deň. Vývojári, ktorí zasahovali do kódu, ktorý neprešiel procesom zostavovania, by mali svoj kód bezodkladne opraviť. Tiež sa zvykli používať aj akési formy trestov za takéto chyby, ako dostať na druhý deň nevýhodnú funkciu v tíme (napríklad tester ostatných modulov), alebo zaplatiť drobný poplatok.

Programátori v Netscape pracovali podobne. Použili softvér (Bonsai a CVS/Tinderbox), ktorý v prípade neúspechu kompilácie a zostavovania určitej časti zdrojového kódu zasielal správy vývojárovi, ktorý podľa záznamu posledný pracoval s daným kódom. Ako hovorí Lloyd Tabb z Netscape-u: *Mohlo ísť o e-mailové správy zo znením napríklad: „Hey what we`ve build today is a piece of garbage. Could anybody have broken this?“ V preklade: „Hej, to čo dnes chceme zostaviť je kus šrotu, pokazil to niekto?“. Tieto správy boli posielané zodpovednému vývojárovi, alebo skupine, kým daný problém niekto neodstránil. Bola to pre vývojárov nepríjemná záležitosť, ale docielil sa tak relatívne krátky čas vývoja.*

Finálna stabilizácia a release

Dodanie finálnej verzie softvérového produktu predpokladá prioritizáciu opráv nájdených chýb a opravenie čo najväčšieho počtu vážnych chýb. Podľa možnosti tak, aby sa opravením nepridali do kódu ďalšie iné chyby. Mnohí softvéroví inžinieri vrátane ľudí z Microsoft-u a Netscape-u to nazývajú finálna fáza odstraňovania chýb („debagovacia fáza“) a stabilizačná fáza. Slovo „stabilizovať“ znamená dať vývojárom dostatočnú slobodu, aby mohli v projekte opraviť čo najviac najhorších a najdôležitejších chýb (niekedy je prakticky nemožné opraviť všetko, preto sa chyby opravujú podľa dôležitosti). Odstránením hlavných chýb by sa mal produkt dostať na stabilnejšiu úroveň. Takýto stabilizovaný produkt potom možno vydať.

Záver

Tento článok opisuje proces vývoja softvéru, v dynamickom, rýchlo sa meniacom prostredí Internetu s vysokým stupňom neistoty. Proces sa teda tiež stal dynamickejším avšak nie absolútne živelným. Prístup míľnikov umožňuje nestratiť nad procesom kontrolu a zároveň umožňuje realizovať a dopĺňať zmeny, ktoré si prostredie vyžiada počas realizácie projektu. Dve nezávislé, konkurenčné firmy Microsoft a Netscape, zaoberajúce sa vývojom softvéru v prostredí Internetu používajú podobné procesy vývoja softvéru. Tieto procesy možno zovšeobecniť a zahrnúť pod spoločný názov „Synchronizuj a stabilizuj“.

Z iných, klasickejších procesov je blízky procesu prototypovania, nesie však svoje špecifiká, čo ho robí samostatným procesom. Z modernejších procesov možno nájsť podobnosť s dokonalým, či extrémnym programovaním (extreme programming), kde sa nachádza taktiež viacero akýchsi míľnikov, v ktorých sa zostavuje produkt, sú však častejšie a je ich tým pádom viac. Rozdiel funkčnosti produktu v jednotlivých

míľnikoch pri extrémnom programovaní je samozrejme kvôli vyššej frekvencií ich výskytu menší ako pri synchronizuj a stabilizuj. Niekedy totiž môže ísť len o pridanie jednej funkcie produktu. Vďaka existencií týchto míľnikov, či bodov čiastkového vydávania produktu sa však nestráca kontakt so zákazníkom počas vývoja, pretože jednotlivé vydania sa u neho overujú a konzultujú, čo ale platí aj pre proces synchronizuj a stabilizuj.

Na záver treba povedať, že nejde o nejakú nepraktickú teoretickú štúdiu, ale že ide o sledovanie praxe a na základe praktických skúseností následné popísanie teórie. Znamená to, že tento proces je používaný a efektívny, čo dokazuje aj fakt jeho využitia spomínanými nezávislými firmami v praxi.

Použitá literatúra

[65] M.A. Cusumano, D.B. Yoffie: Software Development on Internet Time. Vol.32, No. 10, 60-69, October 1999.

[66] M.A. Cusumano and R.W. Selby, Microsoft Secrets" How the Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People, Free Press-Simon & Schuster, NY, 1995, pp. 187-326.

[67] M. Bieliková, Softvérové inžinierstvo, princípy a manažment, ISBN 80-227-1322-8, STU Bratislava.Softvérový proces a projekt, 2000.

[68] Jim McCarthy, Softwarové projekty, ISBN 80-7226-164-0, Computer Press, 1999.

Open Source

Roman Filkorn

Postupnou analýzou hľadáme dôvody, skrývajúce sa za kvalitou Open Source vývoja softvérových produktov. Bližšie sledujeme hľadiská kto, čo, ako, za čo a prečo- porovnáme ju s klasickými metódami vývoja. Načrtávame základné klady a zápory tejto filozofie, výhody a obmedzenia a pokúšame sa určiť jej úlohu vo formovaní budúceho informačného sveta.

Úspech kvality

V poslednej dobe fascinuje svet softvérového vývoja prudký štart produktov označených ako *Open Source* v oblastiach, ktoré boli nepísanou doménou komerčne vyvíjaných produktov. Za všetky spomeňme dve vlajkové lode tohto smeru- projekty Linux (tvorba jadra operačného systému) a Apache (vývoj web-servera).

Čím sa dokázali presadiť programy, ktoré nezastrešuje známa firma z danej oblasti, ale vznikli a rastú v "podhubí" softvérového sveta?

Nájsť podstatu úspechu nevyžaduje dlhú úvahu. Vlastnosť, ktorá sprevádza nasadenie softvérových produktov z naoko rizikového prostredia neexistencie jasného majiteľa, môže byť jedine *kvalita*.

Pokúsme sa hneď v úvode nášho spoločného hľadania definovať základné charakteristiky metódy, ktorá priviedla na svet tieto kvalitné produkty. Základnou črtou je *tvorba na sieti* Internet. Sieť nie je použitá len ako doplnok s obrovskými možnosťami, ale tvorí integrálnu súčasť, chrbticu vývoja produktov Open Source. Široká a *voľná distribúcia zdrojových kódov* otvára možnosti pre zapojenie sa (prakticky) neobmedzeného množstva účastníkov (používateľov, spolutvorcov)- neobmedzuje ich tajomstvo ani *cena* programov.

Ak by nám stačila stručná, populárna formulácia fenoménu Open Source (s dovolením čitateľa nebudem tento pojem prekladať), môžeme v tomto momente skončiť. Na hlbšie pochopenie a vysvetlenie však potrebujeme analyzovať viacero otázok- kto, čo, ako, za čo a prečo; tak isto by sme mali porovnať metódu Open Source, zistiť jej klady a zápory a zamyslieť sa, čo od nej môžeme v budúcnosti očakávať.

Kto – Softvérová komunita

Hoci by sme mali hovoriť o moderných trendoch v oblasti tvorby softvérových systémov, samotný proces, ukrývajúci sa za označením Open Source, existuje už od vzniku siete Internet.

Sieť Internet (resp. jej predchodca ARPANET) na začiatku svojej existencie spájala okrem rôznych štátnych inštitúcií aj vysoké školy (univerzity) v Spojených štátoch. Keďže v tej dobe ešte neexistovala široká ponuka programov a nástrojov na praktické použitie, tvorcovia prvých lastovičiek si vymieňali okrem programov aj zdrojové kódy- v univerzitnom prostredí neexistovala komerčná súťaž o čo najväčší zisk a používateľ si mohol relatívne jednoducho získaný program prispôsobiť. Jedinou hodnotou, ktorú tvorca získal, bolo označenie *hacker*. Pojem, ktorého preklad nie je jednoduchý, ale význam by mal byť úplne jasný. V prácach Erica Raymonda [1,2,3] je hacker chápaný ako človek so značnou *reputáciou*, *prestížou*.

Táto filozofia sa dostala do pozadia v období mohutného rozšírenia sa osobných počítačov bez pripojenia sa k (medzinárodnej) sieti. Obdobie charakterizuje nástup komerčných softvérových spoločností, ponúkajúcich programové balíky priamo na stôl. Na jednej strane prinášajú so sebou rozšírenie ponuky, na druhej strane ale viditeľne klesá kvalita programov pre širokú verejnosť. Pokles kvality je možné hľadať v komerčných tlakoch na firmy, klasických manažérskych postupoch, aplikovaných do prostredia bez existencie ujasnených pravidiel a zákonitostí a v postupnom prevládnutí formy (marketingu a reklamy) nad obsahom (kvalitou a spoľahlivosťou).

Znovuobjavenie sa softvéru s označením OS (Open Source) je možné až po existencii základnej podmienky: verejne prístupné a prudko sa rozširujúce pripojenie k medzinárodnej sieti Internet. Takto sa dostávame do bodu, keď sa znova prejavuje istá potreba- nie potreba *nejakého*, ale *kvalitného* softvéru (jeho nízka cena je až druhoradý faktor). Možnosť konkurovať komerčným produktom vzhľadom na náročnosť tvorby tak rozsiahlych aplikácií len kopíruje veľkosť siete Internet- čim viac zdrojov je k dispozícii, tým je väčšia možnosť uspieť.

Okolo siete Internet (znova) vzniká *softvérová komunita*, ktorej základnou motiváciou je potreba kvalitného softvéru a ktorej jediným priamym ziskom je prestíž. Dobrovoľná práca a jej verejná ponuka charakterizujú *darčekovú kultúru* ([3], preklad [4])- necení sa množstvo práce, ale kvalita práce. Dobrovoľným pripojením svojej práce a intelektu k tvorbe projektu v danej komunite sa potenciálny používateľ stotožňuje aj s fungovaním vzťahov a nepísanými zákonmi komunity- Open Source sa tak nestáva metódou, ale *filozofiou* tvorby softvérových produktov.

Čo – Open Source softvér

Špecifická vývojová komunita (v kontraste s "platenými" pracovníkmi softvérových firiem) si vytvorila svoje vlastné pravidlá. Výsledkom jej práce je *Open Source Softvér (OSS)*, pre ktorý platí: [6]

1. Voľná distribúcia a redistribúcia produktu.
2. Voľný prístup k zdrojovému kódu.
3. Povolená zmena takto distribuovaného softvéru.
4. Zachovanie integrity autorovej práce.
5. Zákaz diskriminovať osoby ani organizácie.
6. Zákaz obmedzenia použitia pre ľubovoľnú oblasť nasadenia.
7. Licencia nesmie byť špecifická k produktu a nesmie poškodzovať iný softvér.

Ak chceme hľadať dôvody, prečo je OSS kvalitnejší než iné konkurenčné produkty, mali by sme začať pri spomenutých pravidlách. Komerčne vyvíjaný softvér sa odlišuje najmä v bodoch 1 až 3 (a čaistočne 7). Ostatné sa vzťahujú k všeobecnejším pravidlám spoločnosti (nediskriminácia, ochrana autorských práv, nekalá súťaž) a až na formu a dosah naplnenia sa neodlišujú od pravidiel pre (takmer) každý softvér.

Hodnota, skrytá za úspešnými projektmi metódy Open Source, je teda determinovaná prvými tromi pravidlami. Ich analýza nám môže pomôcť:

- Voľná distribúcia produktu súvisí so vzdaním sa peňažnej hodnoty za poskytnutý produkt a jej priamym dôsledkom je široké a jednoduché rozšírenie sa produktu medzi potenciálnych používateľov. Od tých sa primárne očakáva používanie programu a odhalenie a zverejnenie nových chýb.
- Voľný prístup k zdrojovému kódu vnáša do procesu novú dimenziu. Používatelia už nie sú slepí (black-box), majú možnosť nahliadnuť do zdrojových kódov a chybu opraviť. Každý používateľ sa môže stať spolutvorcom (metóda Open Source to skoro až vyžaduje), projekt tak môže získať tvorivú základňu s vysokým spoločným potenciálom.
- Pri zmene je možné ju aj distribuovať, oznámiť ostatným svoje doplnenia a opravy. Nepísaným pravidlom je kontaktovať najmä svoj zdroj a zabezpečiť tak oznámenie chyby do integračného strediska.

Predpoklady a základné charakteristiky popisovanej filozofie cítime už z formálnej definície výsledného produktu. Pokiaľ v komerčných firmách bežnými metódami kladieme dôraz na špecifikáciu a návrh (odstránenie chýb čo najskôr) a záverečné testovanie už neprebehne v dostatočnej šírke, komunita okolo dobrovoľného projektu prechádza fázami od vytvorenia prototypu (návrh a implementácia autora projektu), prvotného širokého vyhľadávania chýb, cez spoluprácu na špecifikácii, návrhu a implementácii ďalších verzií až po spoluriadenie a spolurozhodovanie na veľkých a úspešných produktoch. V každej fáze zabezpečuje rozsiahlu nezávislú a neovplyvniteľnú kontrolu vlastnej práce.

Ako – Fungovanie Open Source

Na vývoji produktu v prostredí medzinárodnej siete Internet sa spolupodieľa široká základňa dobrovoľných spolupracovníkov. Títo sa z používateľov, nachádzajúcich chyby v novom programe, postupne menia na spolutvorcov daného programu: participujú na odlaďovaní, na špecifikácii požiadaviek a môžu sa (vzhľadom na prístupnosť k zdrojovým programom) zapojiť aj do tvorby samotného kódu.

Aby takýto proces mohol vôbec začať, musí vyvíjaný produkt spĺňať základné podmienky: *distribuovateľnosť* (možnosť rozdeliť pri väčšom rozsahu medzi viacerých tvorcov) a *modularitu* (nevyhnutná pre distribuovateľnosť). Samotný vývoj je *paralelný* (mnohí ľudia pracujú často na tej istej funkcionalite), *inkrementálny* (postupne sa pridávajú nové funkcionality, neexistuje úplná špecifikácia), *iteratívny* (počas tvorby vzniká niekoľko iterácií (už stabilných verzií)).

Najväčšiu silu skrýva OS vo fáze odlaďovania chýb a opráv. Tu sa v silnej miere presadí *peer-review*, priama kontrola metódou white-box širokým množstvom spolupracovníkov, teda možnosť hľadať zdroj chyby priamo v zdrojových kódoch programu. Ak dotyčný používateľ (spolutvorca) nevie sám danú chybu odstrániť, rozpošle požiadavku do komunity a vždy sa nájde niekto, komu je riešenie jasné. Každá chyba (ak nepredstavuje zásah do hĺbky, návrhu systému) môže byť odstránená v priebehu hodín, najviac dní.

Potenciálne úspešný projekt v komunite OS musí prejsť niekoľkými krokmi. Na začiatku autor špecifikuje problém, vychádzajúc z vlastných potrieb, ktorého riešenie ešte neexistuje na sieti. Musí vytvoriť prvý prototyp a zaujať ostatných ľudí v societe: musí predviesť funkčnosť, ktorá aj im chýba, alebo musí aspoň návrhom a perspektívou prilákať pozornosť ostatných. Takto sa vytvorí okolo programu skupina používateľov, postupne spolutvorcov. Riadená je pomocou mailing listov alebo news serverov na sieti Internet a pomáha autorovi (vlastníkovi projektu) v ďalšom napredovaní.

Hierarchická riadiaca štruktúra okolo OS projektov sa nazýva *benevolentná diktatúra.* Vlastník projektu, pri presiahnutí jeho možností, deleguje právomoci najbližším spolupracovníkom a rozdelí tak prácu na viacero ľudí (s jeho hlavným slovom- diktatúra). Ak on sám z nejakých dôvodov nebude môcť pokračovať v práci na projekte, určí svojho nástupcu (prípadne si ho skupina zvolí sama).

Dôležitým faktorom, sledujúcim postup projektu, je časté šírenie nových verzií- hlavná úloha vlastníka projektu (integrátora). Jedine tak je možné zaručiť kontrolu chýb, udržať motiváciu a riadiace postavenie v skupine okolo tvorby projektu, získavať bližšie a nové špecifikácie požiadaviek (za každou je skutočný používateľ) a analyzovať a spracovávať kvalitné návrhy svojich spolupracovníkov.

Nedostatočné naplnenie niektorého predpokladu môže viesť k neúspechu. Nezáleží na prostriedkoch, ktoré sme schopní do projektu vložiť- ak o ňom zle informujeme, nezaujmeme a nezískame dobrovoľných spolupracovníkov, je naša snaha odsúdená na zánik. Aby projekt mohol byť úspešný, musí vystihnúť a potenciálne pokryť potreby širokého okruhu používateľov.

Za čo – Financovanie Open Source

Vývoj a predaj softvéru je v súčasnej spoločnosti špecifickým typom produkcie. Napriek tomu, že je širokou verejnosťou pokladaný za bežný priemyselný tovar (kde platíme za hotový výrobok a najmä za konkrétnu kópiu), je podstata ceny softvéru skrytá niekde inde (náklady na údržbu sú neraz aspoň také, ako na vývoj; vytvorenie kópie je takmer bez nákladov).

Podľa [2] je v súčasnosti len 5% softvéru určeného na priamy predaj. Ostávajúcich 95% je vytváraných priamo vo väčších spoločnostiach (najmä banková a poisťovacia sféra) samostatnými vývojovými oddeleniami. Softvér šitý na mieru má veľmi nízku možnosť prenesenia do iných podmienok a nie je možné ho prakticky predávať.

Komerčne vytvárané produkty s označením Closed Source (predaj hotových programov v ich výslednej podobe bez možnosti nahliadnuť do zdrojového kódu) sú ocenené najmä na základe nákladov a očakávaného zisku tvorcu. Následná údržba a služby zákazníkom sú v bežnej praxi pokrývané poplatkami len čiastočne a ich financovanie zabezpečuje úspešný predaj niektorého produktu. V prípade zániku firmy okamžite klesá hodnota ním vyvíjaných produktov- zákazník teda platí v očakávaní budúceho servisu od predajcu.

Tvorba a ponuka softvéru je v princípe poskytovaním služieb. Zákazník by mal platiť za vytvorenie služby len minimálne a cena by mala byť odvodená od poskytnutia služby (jej kvality a pod.). Normálnym fungovaním trhového princípu by poplatky mali zahŕňať údržbu a služby zákazníkom. V opačnom prípade sa hlavným cieľom predajcov stane nie kvalita (a zisky z nej plynúce), ale existencia produktu- vytvoriť a čo najrýchlejšie speňažiť bajty svojej práce a predávať ako nové produkty aj opravené verzie (s pridaním istej, často pochybne potrebnej hodnoty) starších produktov.

Filozofia Open Source sa vydala cestou poplatkov. V darčekovej kultúre nejde o zisk za účelom prežitia, ale o pokrytie režijných nákladov a prípadnú motiváciu menej atraktívnych produktov a činností. Časť ziskov z predaja sa používa na organizačnú podporu centier vývoja úspešných projektov. Možné scenáre spoplatňovania OSS načrtávame v nasledujúcich bodoch: [2]

- Open Source klient podporuje predaj Closed Source serverov. Model je na hranici medzi predajom za poplatky a predajom produktov (napr. Netscape Mozilla).
- Hardvérové ovládače. Zverejnením dokumentácie môže predajca hardvéru získať širokú podporu operačných systémov a rozšíriť tak svoj predaj.
- Predaj služieb a pridanej hodnoty je najčistejšou formou poplatkového predaja (napr. Red Hat nepredáva OS Linux ale službu, ktorou skompletizovali operačný systém, vytvorili dokumentáciu a jednoduchú inštaláciu).
- Predaj doplnkov, od tričiek až po kvalitnú profesionálnu dokumentáciu (napr. O'Reilly).
- Predaj Closed Source a poskytnutie zdrojových kódov s licenciou, ktorá obmedzuje voľnú distribúciu na isté časové obdobie. Produkt je možné si

prispôsobiť a náklady na údržbu môže prebrať verejná komunita (napr. Aladdin Enterprises' Ghostscript).

Poslednou alternatívou, do ktorej vkladá nádej komerčný svet a môže sa uplatniť aj ako model pre OS, je prechod na sieťové spracovanie- NetPC. Používateľ bude platiť jedine za používanie softvéru a bude si môcť vybrať na základe poskytovaných služieb a poplatkov poskytovateľa kompletnej softvérovej platformy.

Prečo – Porovnanie Open Source

Porovnanie princípov Open Source a klasických metód zúžime len na základné modely vývoja: vodopádový model, V model, špirálový model, prototypovanie. Ich spoločnou charakteristikou je snaha o čo najpresnejšie zdokumentovanie jednotlivých fáz vývoja a zabezpečenie ich viditeľnosti (čo je úplne zrejmé pri vodopádovom, ale problematickejšie pri prototypovaní).

Väčšina modelov začína špecifikáciou požiadaviek (prototypovanie môže slúžiť na ich overenie a teda špecifikácia môže byť výstupom prototypovania). Nasleduje široká analýza úzkym kruhom skúsených analytikov a návrh riešenia. Kvalita produktu je determinovaná už prvými procesmi- čím kvalitnejšia analýza, tým lepší môže byť návrh, aký dobrý je návrh, len tak dobrá môže byť implementácia (a súvisiaca integrácia). Zlom nastáva pri testovaní- to musí byť vykonané v rovnakej miere nezávisle na (pomýšľanej) kvalite predchádzajúcej práce- až výsledky testovania teda preukážu ozajstnú kvalitu vytvoreného produktu.

Životný cyklus projektu Open Source je iný. Začína úzkou špecifikáciou a implementovaním v malom rozsahu (zväčša autor samotný, resp. malý tím). Následne sa produkt aj so zdrojovým kódom rozšíri medzi používateľov a začína najväčšia fáza tvorby- neustály prísun opráv, zmien, návrhov a špecifikácií bez zjavného systému a striktnej dokumentácie vedie na inkrementálny a iteratívny vývoj. Navonok pôsobí tvorba programu živelne- možno práve preto je potrebná (benevolentná) diktatúra jeho riadenia.

Pri podrobnejšom opise je postup práce nasledovný: prvotný prototyp (napr. vodopádovou metódou) vytvorený úzkou skupinou ľudí, rozšírenie a motivácia používateľov (budúcich spolupracovníkov) a fáza rozsiahleho testovania a dolaďovania. Z tej sa projekt transformuje na paralelnú tvorbu (nedokumentovanej) špecifikácie, poskytovanie návrhových zmien a distribuovanú implementáciu pod stálym tlakom kontroly (rýchle a časté vydávanie nových verzií). Môžeme povedať, že Open Source softvér nie je systematicky budovaný (väčšina klasických modelov), ale postupne rastie s komunitou dobrovoľných používateľov (spoluautorov).

Aké sú teda základné výhody a nevýhody Open Source modelu? Začnime zápornými stránkami (a prípadnou obhajobou):

- *Code and fix [9] postup práce, neexistencia fázy analýzy a návrhu.* Najdôležitejšiu časť práce na prvých dvoch fázach tvorby softvérového systému klasickými metódami vykoná autor projektu, neskôr podlieha procesu peer-review. Je

pravdou, že projekt nemá dokumentovanú špecifikáciu a analýzu, ale z tohto faktu nehrozia žiadne negatívne dôsledky- keďže ďalšie fázy tieto dokumenty nevyžadujú. Dobrovoľnosť spolupráce zabezpečí tie a len tie požiadavky, ktoré používatelia naozaj potrebujú. Návrh systému je v kompetencii vlastníka (diktátora) projektu a jeho kvalita je jedným z motivačných faktorov spolupráce.

- *Overené riešenia [7] (napr. Linux ako nasledovník Unixu)*. Len ťažko možno súhlasiť s výrokom, že nové a originálne nápady nevzniknú v širokej komunite (skúsených) softvérových inžinierov. Odvažujeme sa dokonca tvrdiť opak- práve peer-review a dobrovoľná spolupráca širokej komunity vnáša do projektov omnoho viac originálnych vylepšení ako myšlienky úzkej skupiny pracovníkov ľubovoľnej firmy.
- *Neefektívnosť [7] pri paralelnom nasadení toľkej ľudskej kapacity*. Pravdepodobne je najmä vo fáze odlaďovania chýb istá nadpráca, ale tá je vo veľkej miere odstraňovaná rýchlym a častým vydávaním nových verzií a pozitívne sa prejavuje vo výslednej kvalite produktu. Vo fáze implementácie nadbytočnosť vedie ku konkurenčnému prostrediu viacerých alternatív a vhodné distribuovanie práce a informácii odstraňuje (takmer úplne) nadbytočnosť.
- *Vyžaduje vysokú modularitu, sieťové prepojenie [5]*.Obe pripomienky sú výhradou proti definícií metódy Open Source, ktorá je naozaj obmedzená len na riešenie projektov s modulárnym rozdelením úloh a použitím medzinárodnej siete. V takomto prostredí je ale jednou z najefektívnejších (nakoniec ani iné metódy nemožno použiť v ľubovoľnom prostredí)- v hlavnej miere načrtáva najmä riadenie rozsiahlych medzinárodných projektov.
- *Kontrola projektu je slabšia v dobrovoľnom prostredí*. Open Source projekt nedovoľuje stanovovať žiadne termíny ukončenia, odovzdania, použiteľnosti. Vzhľadom na dobrovoľnosť spolupracovníkov a neexistenciu oficiálneho plánu nie je možné dosiahnuť kontrolovaný postup projektu (nemožnosť represálií, jedine pozitívnou motiváciou).

Z pozitívnych aspektov je možné vyzdvihnúť najmä:

- *Efektívnosť v heterogénnej, rýchlo sa meniacej sieti [5]*. Ako jedna z mála vôbec určuje pravidlá spolupráce v takomto prostredí, aj keď úspešná tvorba projektov by predpokladala stále, homogénne prostredie- ide skôr o obhajobu využitia kapacity siete Internet.
- *Nízke vstupné bariéry, práca len na funkcionalitách, ktoré zákazníci požadujú.* OSS je voľne šíriteľný a tak neexistujú prekážky, ktoré by zabraňovali ho využívať. Neexistencia úplnej špecifikácie ale pripúšťa situáciu, keď by nové požiadavky viedli ku kompletne novému návrhu a implementácii- čo by pravdepodobne znamenalo zánik projektu.
- *nízka cena pre používateľa, široká podpora pri problémoch (chybách, inštalácii, ...)*. OSS sa dá získať za takmer nulovú cenu, problematická ostáva dokumentácia (najmä používateľské príručky). Prvotné problémy s nasadením neznámych (slabšie dokumentovaných) technológií ale vyváži takmer okamžitá

oprava objavených chýb a interaktívna komunikácia so skúsenými pracovníkmi (diskusné fóra, mailing listy).

Nie je jednoduché stručne zhrnúť charakteristiky filozofie Open Source, ale väčšinu diskutovaných aspektov sme zatriedili v nasledujúcej tabuľke:

Tab. 1: Filozofia Open Source.

Aspekt	Vlastnosti
proces	distribuovaný, paralelný, iteratívny, inkrementálny
produkt	voľne distribuovateľný (spolu s zdrojovým kódom), modulárny, prispôsobivý, nezávislý od výrobcu, zväčša slabšie zdokumentovaný
riadenie	benevolentná diktatúra, delegovanie právomocí, sieť, peer-review
komunita	dobrovoľnosť, prestíž, darčeková kultúra, medzinárodná sieť, peer-review
inicializácia	vlastná potreba, vytvorenie prvého prototypu, motivácia
dynamika	rýchle a časté nové verzie

Aká je budúcnosť?

Použitie Open Source kladie isté požiadavky- existenciu medzinárodnej siete, modularitu a darčekovú kultúru, otvorenosť zdrojového kódu bráni použitiu v citlivých oblastiach- vojenská technika, väčšina bankových a poisťovacích produktov. Na druhej strane výborne sa hodí na produkciu ostávajúcich piatich percent softvéru pre širokú verejnosť- poskytuje spoľahlivú, výkonnú a lacnú alternatívu komerčným produktom.

Spoločná práca širokého okruhu ľudí v komunite Open Source je motivovaná vlastnou dobrovoľnosťou a záujmom o konkrétny projekt a vedie k ponímaniu práce ako zábavy- čo je cieľom rôznych (umelých) psychologických trikov aj v komerčných softvérových domoch [10]. Prístupnosť OSS napomáha ich nasadeniu najmä vo finančne skromnejších oblastiach- už dnes vychádzajú vyškolení programátori pre OS Linux priamo z prostredia vysokých škôl.

V súčasnosti sme svedkami širokej deklarovanej podpory rôznych komerčných firiem operačnému systému Linux. K silnej sieťovej podpore sa pridávajú základné balíky- kancelársky softvér, multimediálne spracovanie. Projekt Apache už dnes prevádzkuje šesťdesiat percent serverov na Internete a pomer stále rastie, internetový prehliadač Mozilla (pôvodne Netscape Communicator) opäť získava svoju stratenú pozíciu na trhu. Možné presadenie sa Open Source platformy ako poskytovateľa služieb pre

NetPC a komunikačnú spoločnosť budúcnosti otvára tejto "modernej" filozofii dvere budúcnosti dokorán.

Použitá literatúra

[69] Raymond, E. S.: The Cathedraal and the Bazaar, august 1999
http://www.tuxedo.org/~esr/writings/cathedral-bazaar

[70] Raymond, E. S.: The Magic Cauldron, jún 1999
http://www.tuxedo.org/~esr/writings/magic-cauldron/magic-cauldron.html

[71] Raymond, E. S.: Homesteading the Noosphere,
http://www.tuxedo.org/~esr/writings/homesteading

[72] Tarapčík, I.: Tvorba otvorených softvérových systémov, diplomová práca, FEI STU, 1999

[73] Johnson, K.: Towards a Descriptive Process for Open-Source Software Development, Submission for the 2nd Workshop on Software Engineering over the Internet, apríl 1999
http://www.cpsc.ucalgary.ca/~johnsonk/SENG/SENG691/towards.htm

[74] Open Source Homepage, http://www.opensource.org

[75] S. McConnel: Open-Source Methodology: Ready for Prime Time?, IEEE Software, júl/august 1999, 6-8

[76] T. Bollinger, R. Nelson, K. M. Self, S. J. Turnbull: Open-Source Methods: Peering Through the Clutter, IEEE Software, júl/august 1999, 8-11

[77] G. Wilson: Is the Open-Source Community Setting a Bad Example?, IEEE Software, január/február 1999, 23-25

[78] J. McCarthy: Softwarové projekty, Computer Press, Praha, 1999, ISBN 80-7226-194-0

Certifikačné testovanie

Jaroslav Hanúsek

Procesy testovania tvoria v súčasnej dobe neoddeliteľnú zložku životného cyklu každého softvéru. Ich význam jednoznačne vyplýva z povinnosti dodržania špecifikácie a uspokojenia požiadaviek zákazníka. Nie všetky chyby sú však testovaním odhaliteľné a stávajú sa aj situácie, že nevieme ani povedať, čo vlastne chyba je. Pre vysoko spoľahlivé systémy (armáda, kozmický výskum, lekárske prístroje...) objavenie neočakávanej chyby až počas behu systému môže mať katastrofálne následky - straty na ľudských životoch, obrovské materiálne a finančné straty – straty oveľa väčšie ako náklady na dôkladné testovanie. Nasledovný článok sa snaží priblížiť niekoľko metód, ktoré môžu rapídne zvýšiť bezpečnosť a spoľahlivosť.

Položme si na začiatok jednoduchú otázku: Čo je to softvérové testovanie ? V literatúre sa môžeme dočítať, že: „*Testovanie zahŕňa riadené procesy systému alebo aplikácie a následné vyhodnocovanie výsledkov.*“ [3] Tieto riadené procesy by mali zahŕňať navodenie takých stavov systému, aby sa dala overiť jeho správna funkčnosť a aby sa odhalili možné neželané stavy, do ktorých sa systém môže dostať. Väčšina takto formulovaných procesov testovania je orientovaná na detekciu, resp. prevenciu pred vznikom neželaných stavov, čo má svoje korene vo finančnom pohľade na systém.

Odstránenie všetkých chýb z programu je nemožné, preto treba predpokladať, že ak nejaká chyba v systéme je, tak sa zákonite prejaví. Z tohto dôvodu je na mieste pouvažovať o analýze možných strát a o ich minimalizácii. Na tento problém sa však treba pozrieť aj z druhej strany – čím viac analyzujeme a testujeme, tým viac zvyšujeme náklady na vývoj produktu. Má vôbec význam uvažovať o dôkladnom testovaní, keď celkové náklady na jeho realizáciu presiahnu straty, ktoré by vznikli prípadným výskytom chyby?

Investícia do testovania = zbytočná investícia?

V dnešnej praxi sa stretávame pomerne často so situáciou, že softvérový produkt je zavedený do praxe napriek tomu, že sú známe jeho chyby a jeho „nespoľahlivosť“. Táto situácia väčšinou vzniká ako rozhodnutie manažmentu projektov, kde sa porovnávajú náklady s možnými stratami. Z takéhoto princípu však nemôžme

v žiadnom prípade hovoriť, že dostávame spoľahlivý produkt. A to aj napriek tomu, že systém prešiel testovaním a schválením uvedenia do praxe. Pokiaľ však máme na starosti tvorbu vysoko spoľahlivého systému, resp. softvérovej aplikácie, ktorá bude neoddeliteľnou súčasťou takéhoto systému, musíme zvoliť iný postup.

Pre systémy, kde výskyt chyby znamená obrovské škody (vojenské systémy, zdravotnícke systémy, projekty pre výskum vesmíru, letecké systémy, systémy pre prácu v extrémne nebezpečných podmienkach a pod.), treba zaviesť testovacie procesy v oveľa väčšom rozsahu, ako pri bežných aplikáciách. Otázka financií by mala byť až druhoradá. Uveďme niekoľko príkladov, kedy softvérový systém spôsobil obrovské škody:

- V októbri 1999 bolo prerušené spojenie so sondou Mars Orbiter, pričom nasledovala jej pravdepodobná havária. Škoda NASA v tomto prípade predstavovala $125 000 000 (v prepočte asi 5,5 miliardy Sk). Príčina tejto straty bola odhalená – jedna časť systému používala metrické a druhá britské merné jednotky.
- Zlyhanie softvéru malo na svedomí aj „najlacnejšiu“ nehodu v dejinách kozmických štartov z Cape Canaveral – vypustenie vojenského satelitu za 1,2 miliardy dolárov (55 mld. Sk) skončilo už pri štarte. Táto nehoda vyvolala požiadavku na komplexné preprojektovanie integrácie a testovania systémov.
- V januári 1998 chyba v softvérovom systéme spôsobila, že istá telekomunikačná spoločnosť „zabudla“ účtovať 400 000 diaľkových hovorov. Táto chyba bola odhalená náhodou – až po otázkach zákazníkov spoločnosti na výšku ich účtov.
- V roku 1983 v Sovietskom zväze došlo k pomýleniu sa vojenského systému, ktorý mal na starosť filtráciu falošných raketových útokov. Katastrofa spustenia nukleárnej vojny bola zažehnaná, keďže službukonajúci veliteľ „*...mal vtipný pocit, že ide asi o falošný poplach.*“

Pre systémy, ktoré si vyžadujú vysokú spoľahlivosť, je treba zabezpečiť také procesy testovania, ktoré nedovolia výskyt žiadnej neočakávanej udalosti spôsobujúcej škodu. Keďže takto položenú požiadavku by bolo veľmi ťažko realizovať, zamerajme sa radšej na riešenie otázok:

- Čo budeme považovať za „otestované“ ?
  - Kedy môžeme považovať testovanú aplikáciu za vysoko spoľahlivú?
  - Má význam dodatočné testovanie pre prax?
  - Oplatí sa investovať do dodatočných testovacích procesov?

Na posledné dve otázky existuje jednoznačná odpoveď vyplývajúca z uvedených príkladov, kde softvérové systémy spôsobili obrovské škody. Na prvé dve otázky hľadajú odpoveď procesy, ktoré môžeme nazvať *Certifikáciou softvéru* [1].

Klasické metódy

Aby sme vôbec mohli hovoriť o dodatočnom testovaní, treba si uvedomiť, že certifikačné procesy prichádzajú na rad až po ukončení „bežného“ testovania. V tomto význame budeme považovať za bežné testovanie klasické metódy:

- Metóda čiernej skrinky – procesy, pri ktorých sa sleduje spoľahlivosť systému z pohľadu funkčnosti.
  - Metóda bielej skrinky – procesy, kde sa sleduje správna vnútorná logika aplikácie.
  - Inkrementálne testovanie, testovanie komponentov (integračné testovanie).
  - Testovanie rozhraní (hraničných hodnôt, používateľské rozhrania a pod.) a ďalšie.

Tieto základné testy sú nutnou podmienkou akceptačného procesu, ich zlyhanie si musí vyžiadať opravu softvéru a následné opätovné testovanie.

Čím však chceme mať produkt spoľahlivejší, tým viac rôznych testov musíme vykonať. Slovo „*viac*“ je v tomto prípade veľmi výstižné, lebo presný rozsah testovacieho procesu je ohraničený „*ako veľmi*“ chceme mať softvér spoľahlivý. Preto je vhodné zaradiť do fázy testovania aj niektoré menej štandardné metódy:

Testovanie neočakávaných údajov

V praxi sa stretávame, že používateľské vstupy majú striktné ohraničenia (napr. minimálnu a maximálnu hodnotu). V prípade, že aplikácia akceptuje vstup hodnoty od 1 do 1000, budeme testovať pár správnych vstupov (medzi 1 a 1000), hraničné hodnoty a pár nesprávnych hodnôt (1030, 0). Takéto možnosti získame správnym výberom tried ekvivalencie [7]. Ak však chceme otestovať správnosť, musíme uvažovať aj o nasledovných situáciách: používateľ (človek, sonda, snímač) zadá hodnoty 9999999999999999999999999, 1 (10000 x backspace) 1000, 1000abc0000, ++999 a pod. Softvér aplikácie môže na zadané hodnoty reagovať správne, ale miesto zlyhania budú vstupné zásobníky (input buffers). Veľmi dobrým príkladom je istý operačný systém, ktorý pri zadaní extrémne dlhého prihlasovacieho slova (hesla) reaguje zmrznutím systému s hláškou o neidentifikovateľnej chybe.

Časovače vstupno-výstupných tokov

Predstavme si situáciu, že softvér telefónu reaguje pri vytáčaní cifier telefónneho čísla na presne stanovené časové úseky. Aká bude reakcia, keď používateľ medzi piatou a šiestou cifrou počká trošku dlhšie. Zareaguje softvér adekvátne? (Počká alebo vytočí päťciferné číslo?) Takáto modelová situácia vzniká pri mnohých procesoch (čakanie na správy v sieti, zadávanie časovo závislých údajov a pod.)

Čo keď...

Ako zareaguje softvér, keď vznikne neočakávaná udalosť. Vypíše chybovú hlášku? Ukončí aplikáciu? „Zmrzne“? Ako ovplyvní ostatné súčasti systému? Čim chceme spoľahlivejší softvér, tým viac musíme myslieť na nedokonalosť softvéru, na možné havárie a vplyv na systém ako taký. Snáď netreba zdôrazňovať, že pri vysoko spoľahlivých systémoch haváriu nepovažujeme za prípustný stav ukončenia systému.

Chybné hraničné mierky

Ako už bolo spomínané, každý vstup systému je projektovaný pre istý rozsah hodnôt. Keď vieme, že teplota planéty je v rozsahu –100 do +100 stupňov, implementujeme softvér, ktorý správne zareaguje na tieto hodnoty prichádzajúce od senzorov. Čo však ak príde situácia, že teplota sa bude pohybovať od 50 do 150 stupňov. Môžeme sa vyhovoriť, že podľa špecifikácie nás teploty nad 100 stupňov nezaujímajú? Tu sa presúvame do roviny, kde sa dá diskutovať nie o zlom testovaní, ale zlej špecifikácii. A to je presne podstata rozdielu medzi klasickým a certifikačným testovaním.

Certifikačné testovanie

Certifikácia vo všeobecnosti by mal byť proces, ktorý sleduje kvalitu produktu ako takého, bez ohľadu na procesy, ktorými produkt prešiel vo fáze vývoja. V súčasnej dobe sa často objavuje certifikácia, ktorá vychádza za predpokladu, že *kvalitný proces* produkuje *kvalitný výrobok.* Môžeme preto povedať, že certifikáciou prebieha proces a nie výrobok ako taký.

Pri certifikovaní vysoko spoľahlivého systému je ťažisko na kvalite produktu. Rozdiel medzi testovaním a certifikáciou môžeme definovať nasledovne:

- Softvér prejde testovacími procesmi vtedy, keď splní špecifikáciu (požiadavky na softvér).
- Softvér prejde certifikáciou vtedy, keď nevyvolá nežiadúce stavy systému odporujúce špecifikácii.

Z tejto definície je zrejmé, že je potrebné rozlišovať softvér a celkový systém, ktorého je softvér súčasťou. V tomto ponímaní je za softvér považovaný každý komponent, ktorý je súčasťou iného (väčšieho) systému.

Certifikačný test preto sleduje správanie sa softvéru nie z hľadiska jednotlivých komponentov, ale z vplyvu na systém ako taký. Tu sa vynárajú nasledovné skutočnosti:

- splnenie špecifikácie softvéru nezaručí systému stabilitu a naopak
- dodržanie špecifikácie môže vyvolať chaos na úrovni systému
- správna externá funkčnosť môže skrývať interný chaos

Podstatné pre certifikačný proces je to, že nerozlišuje stavy na spĺňajúce špecifikácie a nespĺňajúce, ale na žiaduce stavy (nepriečiace sa špecifikácii) a nežiaduce. V praxi môže nastať situácia, že zlé správanie softvéru (nie podľa špecifikácie) nespôsobí zlé správanie systému. Horší prípad je však ten, keď správne funkčný softvér vyvolá nežiaduci stav celého systému. Aby sme mohli vyhodnotiť certifikačný test, musíme si zadefinovať, čo je správne a čo nie. V tomto prípade – definuje teda nielen to, ako sa softvér má správať, ale hlavne to, ako sa nesmie správať.

Kedy a kto ?

Tak ako existujú dva pohľady na certifikáciu, tak isto môžeme hovoriť o dvoch pohľadoch na zaradenie certifikačných procesov do procesu vývoja. Prvý prístup definuje testovanie ako jeden celok – certifikačné testy sú súčasťou vývoja. V tomto prípade teda poznajú štruktúru softvéru a môžu pristupovať k testom metódami bielej skrinky. Takto zaradený proces certifikácie môže vykonávať interný odborník, ktorý produkt pozná.

Druhý prístup striktnejšie oddeľuje certifikačné testy. Vykonáva ich v tomto prípade externý odborník, ktorý nemusí poznať procesy, ktorými softvér prešiel počas vývoja (obr.1).

Procesy certifikácie

Jeffrey Voas vo svojej publikácií [1] determinuje tri procesy, ktoré sú vhodné na certifikačné testy:

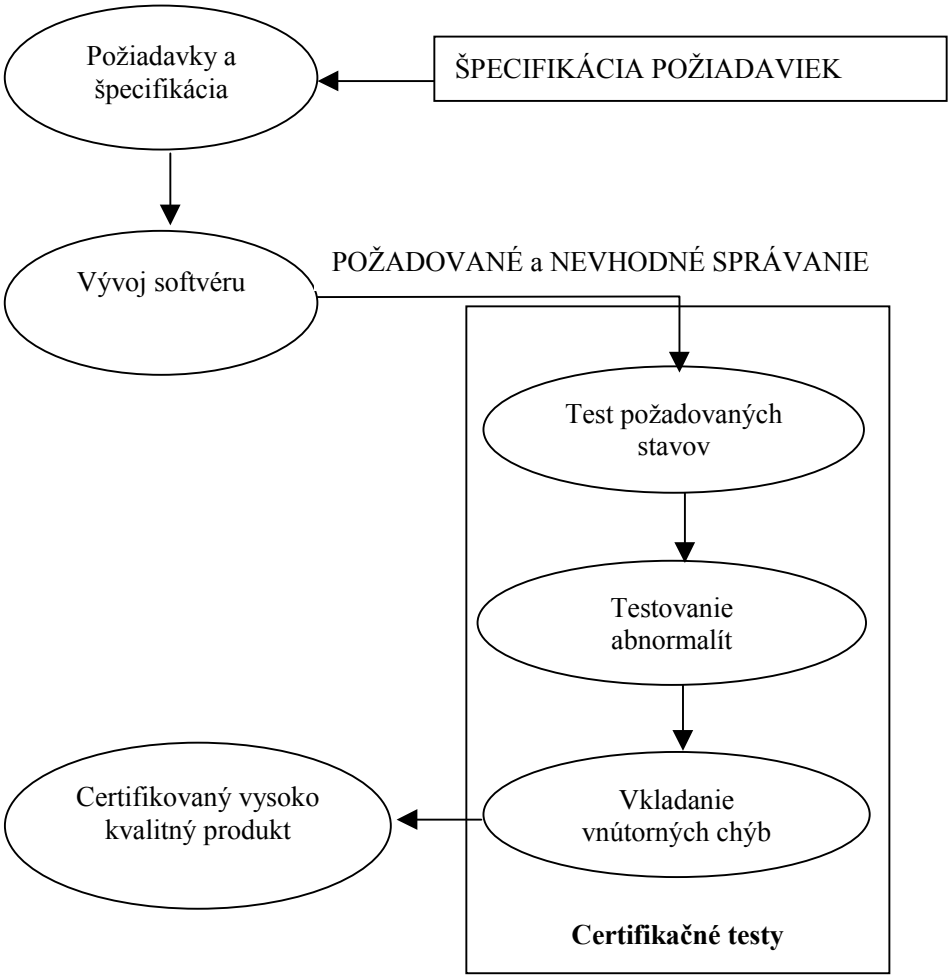
A) testovanie žiaduceho správania sa systému (Desirable-behavior testing)

B) testovanie anomálií (Abnormal testing)

C) Vkladanie chýb (Fault injection)

Testovanie žiaduceho správania

Pri dodržaní požiadaviek na vstupy, systém zostane v stave spĺňajúcom špecifikáciu alebo v stave, ktorý síce nespĺňa špecifikáciu (alebo nie je definovaný), ale nemá negatívny vplyv na systém ako celok. Ide o testovanie typu „čierna skrinka“, pri ktorom sa sleduje spoľahlivosť celkového systému a nie nutne softvéru.



Obr. 1: Priebeh vývoja vysoko spoľahlivého softvéru.

Testovanie anomálií

Inverzia špecifikácie – testovanie opačných (nesprávnych, zriedkavých) stavov a vstupov. Ak je pravdepodobnosť hodnoty na vstupe prakticky nulová, pri bežnom testovaní sa vynecháva (vychádzajúc z pravdepodobnosti výskytu v praxi). Pri abnormálnom testovaní sa takéto hodnoty preferujú.

Väčšina takto odhalených chýb bola dôsledkom viacnásobného výskytu anomálií (NASA).

Vkladanie chýb

Ide o testovanie, ktoré je založené na vkladaní umelo poškodených komponentov do testovaného softvéru. Sleduje sa, ako sa systém správa ako celok. Keďže hovoríme o vysoko spoľahlivých produktoch, musí byť zabezpečená spoľahlivosť za každých okolností. Systém sa nesmie dostať do nežiaduceho stavu napriek tomu, že obsahuje chybný komponent. Táto metóda vychádza z testovania hardvéru, je pomerne náročná a drahá a vyžaduje isté znalosti o stavbe testovaného systému. Prax ukázala, že ide o veľmi účinnú metódu a v súčasnosti tvorí neoddeliteľnú súčasť testov.

Učme sa na vlastných chybách

Tak ako aj v bežných testoch, kvalitu certifikačných testov obmedzuje zvolený rozsah. Čím viac testov (testovacích údajov) sa použije, tým viac je výsledok spoľahlivejší. Čim menej testov použijeme, tým viac ušetríme. Ak si certifikačný tím dovolí podceniť faktor rozsahu testov, jeho reputácia sa poškvrní veľmi radikálnym spôsobom hneď po prvej katastrofe...

Dilema zo začiatku článku sa nerieši ani zavedením certifikačných testov, otázka rozsahu testu bude vždy závisieť nielen od vývojárov, ale aj od finančných manažérov. Položiť si preto otázku, či „*otestovaný softvér bude kvalitným softvérom*," z hľadiska relatívnosti pojmov nemá význam. Rozumné je si ale uvedomiť, že „*dostatočne otestovaný softvér bude vždy len dostatočne kvalitný a len dostatočne spoľahlivý*."

V prostredí systémov, kde sa vyžaduje maximálna spoľahlivosť, by slovíčko „*dostatočne*" malo znamenať : „*maximálne ako sa dá*." Pokiaľ pri bežných aplikáciách môžeme vypustiť produkt aj so známymi chybami, pri vysoko spoľahlivých systémoch takéto niečo nepripadá do úvahy. Akonáhle zlyhá čo i len jeden certifikačný test, systém musí ísť späť do vývoja.

Použitá literatúra

[79] Voas Jeffrey: Certifying Software for High-Assurance Environments, IEEE Software, July/August 1999, vol.14/6, str.48-54

[80] Voas J., McGraw G., Kassab L., Voas L.: A "Crystal Ball" for Software Liability, IEEE Computer 1997

[81] Hower Rick: Software QA and Testing Frequently-Asked-Questions, Part 1, 1996-2000, http://www.netcom.com, aktualizované 2.1.2000

[82] Kaner C.: The Impossibility of Complete Testing, Law of Software Quality Column, Software QA magazine, 1997, http://www.badsoftware.com, aktualizované 9.11.1997

[83] Voas J.: Why COTS Software Increases Security Risks, http://www.isacc.com/presentations/, aktualizované 18.3.1997

[84] Kaner C.: What is a Serious Bug?, 1996, http://www.badsoftware.com, aktualizované 12.10.1997

[85] Bieliková M.: Softvérové inžinierstvo, FEI STU Bratislava, 2000, str.83-91

Záver

Teória o vývoji softvéru rozhodne nie je samoúčelná. Je pravda, že často boli nami popisované procesy vývoja softvéru najprv používané a vyvíjané v praxi, a až potom bola pre ne vytvorená a zosúladená teória. Toto platí napríklad o „open source“, alebo „návrhových vzoroch“, kde pri „open source“ najprv možno až živelne vznikol daný proces vývoja softvéru a až potom sa skutočne popísal. A u „návrhových vzorov“ je priam ich podstatou, že vychádzajú z praktických, mohli by sme povedať empirických skúseností. Takto je to však aj v mnohých iných vedných odvetviach. Aj vedci vo fyzike často prevedú experiment a na základe jeho výsledku sa snažia overiť, alebo získať vhodnú teóriu. Toto by však bolo úplne zbytočné, keby teória len popísala prax a nepriniesla nič nové. Teória by potom bola len akousi dokumentáciou či správou o pozorovanej realite. V skutočnosti je to však tak, že vznikom teórie sa umožní získať lepší pohľad na skúmanú oblasť a v mnohých prípadoch sa čiastkové praktické závery prostredníctvom dobrej teórie spoja ako mozaika v obraz, ktorý presahuje význam jednotlivých praktických výsledkov. Fyzici potom na základe teórií môžu navrhnúť nové praktické experimenty, alebo postupy ku ktorým by sa inak dostali snáď len náhodne. Vo vývoji softvéru to znamená, že teória vytvorí ucelený pohľad na daný proces vývoja a umožní jeho efektívne zlepšenie a optimalizáciu, a môže tak viesť k návrhu nových a lepších postupov v praxi. Toto by sa dalo povedať o „dokonalom programovaní“ (anglicky „extreme programming“), alebo „synchronizuj a stabilizuj“, ale aj o mnohých iných procesoch vývoja softvéru.

Samotný softvér aj jeho vývoj v sebe vždy niesli a pravdepodobne aj ponesú istý faktor živelnosti, ktorý je koniec koncov skôr prínosom ako brzdou vývoja. Napriek existencii tohto nevypočítateľného faktoru veríme, že aj náš prehľad prístupov k vývoju softvéru vedie k zlepšeniu prehľadu čitateľa a je pre neho prínosom a inšpiráciou.

Bc. Marcel Bočkay
Bc. Peter Burger
Bc. Matej Černák
Bc. Roman Ďumbier
Bc. Robert Dvoran
Bc. Roman Filkorn
Bc. Jaroslav Hanúsek
Bc. Peter Jehlár
Bc. Tomáš Milička
Bc. Filip Pucher
Bc. Branislav Šimo

Softvérové inžinierstvo
Etický kódex a procesy vývoja softvéru

Grafická úprava a obálka Peter Jehlár, Tomáš Milička
Vydala tvorivá skupina študentov s názvom GWAIN
Vydanie prvé a posledné
Náklad 15 výtlačkov
Rok vydania 2000
Nepredajné