Chyba a manažment softvérového projektu

Nerozlučiteľný spoločník

TOMÁŠ MINČEFF

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
minceff01@student.fiit.stuba.sk

Abstrakt. Chyba je nerozlučným partnerom akéhokoľvek softvérového projektu. Tento stav je zapríčinený viacerými faktormi. Asi najdôležitejším je osobitosť vývoja softvéru a jeho vlastnosti. Pozrieme sa na tvorcov chýb a ich skúsenosti. Popíšeme chyby z hľadiska ich viditeľnosti a následkov, ktoré z nich vyplývajú. Testovanie patrí k oblastiam, ktoré sa často podceňujú, a preto sa zameriame na dôvody tohto stavu a vzťah testovača a programátora. S chybami priamo súvisí spoľahlivosť softvérového produktu. Ako jednému zo základných modelov správania sa chyby sa budeme venovať modelu Musa. Nakoniec si porovnáme náklady, ktoré platíme za chyby a ako sa mení ich cena v závislosti od viacerých faktorov.

Úvod

Keď sa používatelia rozprávajú o softvéri, s veľkou pravdepodobnosťou sa zaoberajú jeho nedostatkami a najmä chybami. Dôvodom je skutočnosť, že chyby im spôsobujú nejakú škodu, či už ide o čas strávený napr. pri opakovanom zadávaní údajov alebo stratu finančných prostriedkov. Používatelia sa často bezvýhradne spoliehajú na softvér, pretože im uľahčuje prácu. Preto je kvalita a spoľahlivosť softvéru posudzovaná v prvom rade podľa výskytu chýb.

Ani po takmer polstoročí vývoja softvérových produktov neexistuje žiadny postup alebo metodika, ktorá by viedla k bezchybnému softvéru. V tomto príspevku sa pozrieme na príčiny tohto stavu, ako sa postaviť k chybám a možnostiam eliminácie chýb. Neobídeme ani cenu, ktorú platíme za chyby.

Chyba a manažment

Ak by sme chceli explicitne vyjadriť vzťah medzi chybou a manažmentom softvérového projektu, potom cieľom manažmentu projektu je okrem vytvorenia

Manažment v softvérovom inžinierstve, december 2005, s. 1-9.

samotného produktu snaha minimalizovať výskyt chýb a zabezpečiť spoľahlivosť a kvalitu softvéru. Výskyt chýb u používateľa spôsobuje nielen stratu finančných prostriedkov potrebných na ich odstránenie, ale môže poškodiť dobré meno softvérového manažéra a celej spoločnosti.

Chyby nemusia priamo súvisieť len s používateľmi. Niektoré chyby dokonca môžu spôsobiť problémy samotnému dodávateľovi, napr. vtedy ak nie je zabezpečená dostatočná modifikovateľnosť. Z toho dôvodu je potrebné riadiť vývoj takým spôsobom, aby sa chyby v dostatočnom množstve odstraňovali ešte pred uvedením softvérového produktu na trh.

Osobitosť vývoja softvérového produktu

Vývoj softvérového produktu sa výrazne líši od ostatných výrobných činností. V bežných inžinierskych odvetviach, ako napr. stavebníctvo, nie je predstaviteľné, aby sa nejaký projekt dokončil a potom sa otestoval, či nezlyhá a splní svoje funkcie. Hlavnou príčinou je zničenie takmer všetkých zdrojov, ktoré sa použili na jeho vytvorenie. Ďalej ich nie je možné použiť.

Softvérový projekt sa skladá z veľkého množstva netriviálnych modulov, ktoré je možné znovu použiť. V prípade výskytu chyby vo veľkom množstve prípadov stačí upraviť len niektoré jeho časti. Len v malom percente prípadov sa moduly vyradia z činnosti a začnú sa budovať odznovu.

Je možné vytvoriť projekt bez chýb? Takáto možnosť existuje iba v prípade malých systémov, resp. programov, ktorých rozsah je niekoľko funkcií, prípadne tried bez zložitých závislostí. Príkladom je implementácia triviálnych algoritmov, ako sú matematické výpočty, pri ktorých je možné preukázať jeho správnosť.

Autori chýb

Ako povedal Mark Paulk z Univerzity Carnegie Mellon: „Základným problémom s kvalitou softvéru je to, že programátori robia chyby.“ A pritom je úplne jedno, či ide o skúseného programátora alebo o programátora-začiatočníka.

Jedným z hlavných dôvodov, prečo aj skúsený programátor vytvára chyby, je skutočnosť, že je nasadzovaný na riešenie komplikovanejších úloh prinášajúcich so sebou vyššiu zložitosť. Z toho vyplývajú aj vyššie nároky na odhalenie ich príčin ako chyby spôsobené len preklepmi menej skúsených kolegov [1].

No nielen programátori sú zodpovední za chyby, ale aj analytici, návrhári, dokonca aj samotní zákazníci. Mohli by sme povedať, že hlavnou príčinou sú nedostatky v komunikácií a neporozumenie dokumentácii. A to už od samotného špecifikovania požiadaviek. Následkom toho zákazník nedostane to, čo chcel a považuje to samozrejme za chybu. Pritom môžu byť splnené všetky požiadavky vyplývajúce zo zmluvných záväzkov.

Viditeľnosť chyby

Chyba sprevádza softvérový projekt počas celého jeho života. Od samotného zadania projektu cez analýzu, návrh, vývoj, údržbu až po vyradenie z činnosti.

Chyby môžeme z hľadiska viditeľnosti rozdeliť na externé a interné [2]. Externá chyba je ľahko odhaliteľná pre bežného používateľa. Zjednodušene program nevykonáva, čo je definované v jeho špecifikácii a je to jasne viditeľné okamžite po jeho vykonaní (resp. nevykonaní).

Interné chyby, často nazývané skrytými chybami, nemajú efekt pri bežných operáciách. Ležia nepovšimnuté a neodhalené testovaním a dozvieme sa o nich pri vzniku neočakávaných výnimiek alebo pri údržbe produktu, kedy sú odhalené programátorom pri opätovnom prechádzaní kódu.

K interným chybám patrí aj nedodržiavanie štandardov formátovania a pomenovania objektov zdrojového kódu. Takéto chyby spôsobujú problémy pre programátora pri budúcej údržbe softvéru.

Ďalšou formou skrytých chýb je definovanie objektov, ktoré nie sú nikde použité, nesúvisiace komentáre, nevhodné použitie dátových štruktúr, výber neefektívnych algoritmov a pod.

Príklad zo školského tímového projektu

Príkladom projektov, ktoré obsahujú skryté chyby, je riešenie témy RoboCup – nové stratégie na predmete Tvorba softvérového systému v tíme. Cieľom projektov je vytvorenie autonómneho hráča, ktorý je schopný samostatne hrať simulačnú futbalovú robotickú ligu. Problém je v náročnosti vývoja takéhoto hráča, preto sa používajú hráči vytvorení v predchádzajúcich ročníkoch a postupne sa vylepšujú. Napriek tomu, že autori deklarujú modularitu a rozšíriteľnosť, prax je úplne iná.

V snahe dokončiť projekt a odladiť hráča pre turnaj sa programátori sústredili na funkcionalitu. Študenti dobre vedia, že sa s týmto zdrojovým textom už viac nestretnú, preto nemajú motiváciu na lepšie dodržiavanie štandardov. V dôsledku toho, každý nasledujúci tím strávi niekoľko dní až týždňov nad analýzou kódu, aby bol schopný pokračovať v práci.

Pre tieto projekty je typické ponechanie pôvodného kódu v zdrojových textoch, aj keď sa už nepoužíva. Ako argument sa uvádza možnosť znovupoužitia. Priznajme si, že miesto na odkladanie myšlienok nie je zdrojový kód, ale jeho dokumentácia.

Ďalším problémom je nedostatočné používanie komentárov a nezahrnutie posledných a podstatných zmien do finálnej dokumentácie. Za takýchto podmienok je odhalenie chyby veľmi zdĺhavé a menej času sa potom môže venovať zlepšovaniu samotného produktu.

Testovanie

Jednou z hlavných príčin zníženia kvality je podcenenie času potrebného na testovanie. V súčasnej dobe sa zákazník snaží minimalizovať čas potrebný na vývoj softvéru.

Pod týmto nátlakom sa snažia manažéri vyhovieť zákazníkom v snahe získať tieto zákazky. Avšak čas na vývoj nie je možné znižovať pod istú hranicu. Stáva sa skôr pravidlom, že manažéri vidia isté rezervy v dokumentovaní a testovaní. Je pravda, že tieto dve činnosti prakticky priamo neovplyvňujú výsledný softvérový produkt. Toto riešenie je ale veľmi krátkozraké.

Obchádzanie testovania má za následok vypustenie chyby zákazníkovi. Keďže firma poskytuje údržbu pre softvérový produkt, chyba sa vráti naspäť ako reklamácia. Takáto chyba sa napokon musí opraviť. Rozdiel je ale v nákladoch, ktoré v tomto prípade zahŕňajú okrem samotnej opravy aj manažment zmien, manažment verzií a prípadne aj konfigurácií. Tiež si to vyžaduje návrat programátora k staršiemu kódu.

Nedostatočná dokumentácia spôsobuje problémy najmä v etape údržby softvéru. Základným problémom je to, že znalosť systému je uložená iba v mysliach pôvodných autorov. Z toho vyplýva zvýšená komunikácia medzi členmi tímu a ťažšie zaškolenie nových pracovníkov. Takáto situácia je živnou pôdou pre vytváranie nových chýb.

Ďalším možným rizikom je nedostatok pracovníkov vykonávajúcich testovanie a tiež odďaľovanie postúpenia nového kódu na testovanie. V takejto situácií sa môže stať, že vo veľmi krátkom čase je potrebné otestovať príliš veľa funkcionality softvéru. Dôsledkom toho sa stráca podstata testovania, ktorou je odhalenie čo najväčšieho počtu chýb a tým zabezpečiť lepšiu spoľahlivosť. Namiesto toho slúži testovanie len ako súčasť schvaľovacieho procesu.

Netreba podceňovať ani automatizované testovanie, ktorého úloha sa výrazne zvyšuje pri údržbe softvéru. Hlavným dôvodom jeho používania je dosiahnutie vysokej kvality softvéru. Aj pri oprave chýb sa do zdrojového kódu prinášajú nové chyby, ktoré môžu spôsobiť dokonca aj väčšie škody, ako by spôsobila pôvodná chyba.

Úloha testovačov

Okrem samotného času na testovanie hrajú dôležitú úlohu testovači. Táto pozícia sa dlhodobo podceňuje, dokonca títo členovia tímu bývajú považovaní za menejcenných. Je to spôsobené možno aj nižšou kvalifikáciou, najmä v prípade, ak ide o tzv. používateľské testovanie.

Prečo nie sú obľúbení? Ich prácou je hodnotenie softvéru a teda aj samotných vývojárov. Či si to už vývojári uvedomujú alebo nie, k svojmu programu majú istý vzťah. Ak sa hodnotí ich program, potom sú hodnotení aj oni sami. Určite nie je nikomu príjemné počúvať o vlastných chybách. Ak zamestnanca hodnotí riadiaci pracovník, ten ho môže motivovať napr. finančne za dobre odvedenú prácu, prípadne naopak. Vždy je tu možnosť odmeny. Testovač nemá ako motivovať autora kódu a stáva sa tak len „poslom zlých správ“.

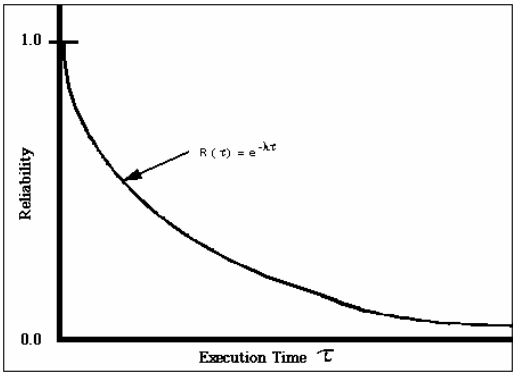
Na druhej strane túto profesiu stále častejšie vykonávajú skúsení programátori, ktorých už vývoj samotných aplikácií omrzel. Nepozerajú sa na softvér ako používatelia, ale ako nezávislí programátori. Majú skúsenosti s vlastnými chybami, poznajú kritické miesta systémov, a preto majú lepšie možnosti odhaliť chyby. Nachádzanie nových chýb sa pre nich stáva „bojovou“ úlohou a odhalenie chyby odmenou.

Spoľahlivosť

Softvérová spoľahlivosť je definovaná ako pravdepodobnosť, že softvér nezapríčiní zlyhanie systému počas špecifikovaného časového úseku pri špecifikovaných podmienkach. Spoľahlivosť je jedna z najstarších metrík používaných pri hodnotení softvéru. Na jej výpočet sa pozrieme pomocou modelovania správania sa chýb.

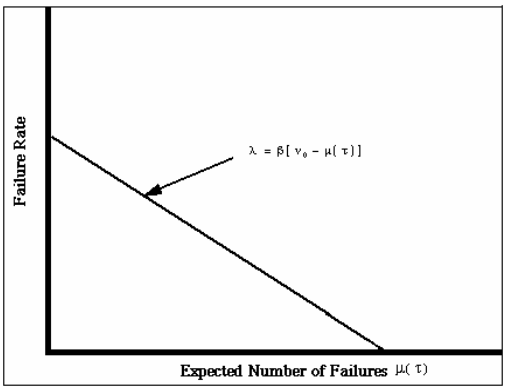
Musa Basic Execution Time Model

Tento model [3] opisuje správanie sa chýb v systéme počas testovania. Pri tomto modeli sa rozlišuje medzi chybou a zlyhaním. Zlyhanie nastáva počas vykonávania programu. Chyba je nedostatok zdrojového kódu, ktorý môže spôsobiť jedno alebo viacero zlyhaní. Predpokladajme, že softvérový systém je nasadený v stabilnom prostredí, pričom sa nemenia používatelia ani nároky používateľov na funkcionalitu systému. Systém sa nemení počas prevádzky. Ak sú tieto požiadavky splnené, potom softvér môžeme modelovať ako systém s konštantnou frekvenciou zlyhania λ, takže množstvo chýb priamo úmerne závisí od času používania systému. Pravdepodobnosť $R(\tau)$, že softvér bude pracovať bez zlyhania, je nepriamo úmerná uvažovanému času. Tento vzťah je možné vidieť na obrázku **Obr. 1**.



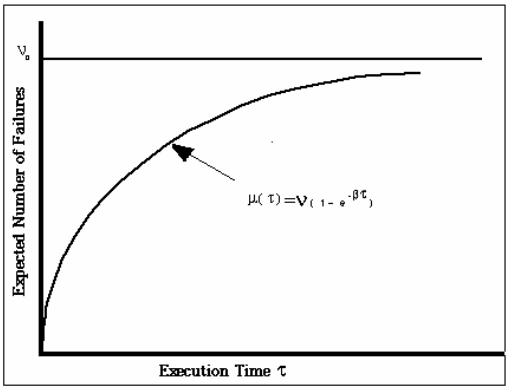
Obr. 1. Závislosť spoľahlivosti od času [3]

Systém sa bude vyvíjať počas testovania za účelom odstránenia pozorovaných chýb, čím sa zvýši jeho spoľahlivosť. Tento model je založený na predpoklade, že všetky chyby prispievajú k chybovosti rovnakým dielom. Takže frekvencia výskytu zlyhaní je klesajúca lineárna funkcia závislá od počtu očakávaných zlyhaní $\mu(\tau)$. Vzťah medzi frekvenciou zlyhania a počtom očakávaných zlyhaní je zachytený na obrázku **Obr. 2**.



Obr. 2. Frekvencia zlyhania v závislosti od očakávaného počtu zlyhaní [3]

Dvoma základnými parametrami tohto modelu sú predpokladaný počet zlyhaní potrebných na odstránenie všetkých chýb – ν_0 a predpokladaný pokles frekvencie zlyhania za zlyhanie – β. Na obrázku **Obr. 3** môžeme vidieť vzťah medzi očakávaným počtom zlyhaní od času.



Obr. 3. Očakávaný počet chýb v závislosti od času [3]

Spoľahlivosť môžeme odhadnúť na základe parametrov, ktoré poznáme z predchádzajúcich etáp vývoja softvéru. Tento odhad je založený na týchto troch parametroch:

- počiatočná frekvencia zlyhaní – λ_0,
- počet chýb na začiatku testovania – ω_0,
- faktor redukcie chýb – *B*.

Frekvencia počtu zlyhaní za chybu ϕ je definovaná ako:

$$\phi = \frac{\lambda_0}{\omega_0} \tag{1}$$

Táto hodnota ϕ je rozdielna od frekvencie zlyhania za zlyhanie β pre vznik chýb počas ladenia programu. Faktor redukcie *B* vyjadruje očakávaný počet odstránených chýb za zlyhanie. Vzťah medzi týmito veličinami sa nachádza v rovniciach 2 a 3.

$$\beta = B\phi = \frac{\lambda_0}{\nu_0} \tag{2}$$

$$\nu_0 = \frac{\omega_0}{B} \tag{3}$$

Keď máme parametre tohto modelu, potom môžeme určiť, ako dlho musí prebiehať testovanie, aby sme dosiahli požadovanú spoľahlivosť. Pre faktor redukcie chýb *B* sa odporúča hodnota 0,955 chyby za zlyhanie.

Náklady na chyby

Už viackrát sme spomínali vzťah medzi chybou a cenou, ktorú za ňu skôr či neskôr zaplatíme. Táto cena závisí od:

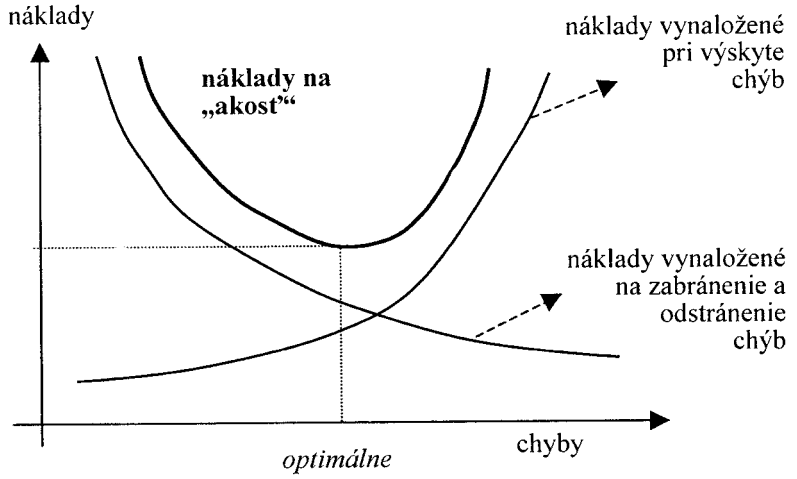
- etapy životného cyklu programu, v ktorom táto chyba vznikla,
- času potrebného na jej odhalenie.

Chyba, ktorá vznikne pri návrhu, môže razantne ovplyvniť nasledujúce etapy projektu najmä v prípade, ak bude potrebné prepracovať doteraz vytvorené dielo. Chyba vytvorená zväčša počas implementácie môže ovplyvňovať produkt ako celok, ale jej odstránenie je zvyčajne iba predmetom jedného modulu.

Čas potrebný na odhalenie chyby opäť spôsobuje zvyšujúce sa náklady na jej odstránenie. Príčinou je napríklad to, že programátor sa venuje modulu, v ktorom nastala chyba, len určitý čas a potom prechádza na riešenie ďalších problémov. Po istom čase si už programátor nepamätá všetky podrobnosti o danom module a je donútený si ich občerstviť, čo stojí nejaký ten čas.

Ďalším aspektom, ktorý treba brať do úvahy, je cena opatrení, ktoré sme vynaložili na elimináciu chýb a efekt, ktorý tieto opatrenia priniesli. Tieto náklady rastú so stúpajúcimi požiadavkami na bezpečnosť, pričom relatívne množstvo odhalených chýb s rastúcimi prostriedkami klesá.

Našou snahou je použitie optimálnych nákladov za podmienky, že zákazník si ich neurčil v zmluve. Optimálne náklady si môžeme odvodiť z grafu na obrázku **Obr. 4** ako maximálny počet odstránených chýb za minimálne investované prostriedky.



Obr. 4. Určenie optimálnych nákladov na kvalitu [4]

Zhodnotenie

Chyba vo všetkých svojich podobách ovplyvňuje nielen naše životy, ale aj softvérové projekty. Aj keď neexistuje žiadny recept na ich odstránenie, nie je vhodné aby sme sa ich báli. Chyby musíme brať ako súčasť riešenia a vytvoriť predpoklady pre ich primerané riadenie s cieľom uspokojiť potreby nielen používateľa, ale aj vývojových pracovníkov, či už ide o programátorov alebo testovačov.

Použitá literatúra

1. Holzmann, G. J.: The Logic of Bugs, ACM SIGSOFT Software Engineering Notes, Vol. 27, No. 6 (2002), 81 – 87.
2. Douce C. R., Layzell P. J.: Evolution and Errors: An Empirical Example, Proc. of ICSM, Oxford, UK (1999), 493 – 498
3. Marciniak J., Vienneau R.: Software Engineering Baselines, Rome, 1996 http://www.dacs.dtic.mil/techs/baselines/reliability.html
4. Bieliková M.: Manažment v softvérovom inžinierstve, 1999

Annotation

Fault and Software Project Management

The fault is inseparable partner of every software project. This state is caused by multiple factors. The most important one is distinctiveness of software development and its characters. We look at authors of faults and their experience. Than describe fault visibility and their consequences. Testing is one of domain, which we often downgrade, so we look at that reason and relationships between tester and programmer. The faults are directly connected with reliability of software product. The Musa is one of the main models of simulation faults in software systems. At the end we compare fault cost and relation between this value and some dependent factors.