Chyby a manažment softvérového projektu

TOMÁŠ TATRANSKÝ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
tomas.tatransky@centrum.sk

Abstrakt. Každý softvér sa počas svojho vývoja musí vysporiadať so vznikom chýb. Vznik chýb je neodvratný pri každej ľudskej činnosti, preto je nutné zaoberať sa problémom vzniku chýb a snažiť sa o ich odstránenie. Tento dokument sa zaoberá rozborom kritických miest softvéru, pri ktorých často vznikajú chyby. Spomína aj rozšírenia softvéru ako systém pluginov a podpora skriptovania, ktoré môžu byť zdrojom chýb, prípadne môžu tvoriť bezpečnostné riziko. Ďalej sa zaoberá možnosťami manažmentu ako predchádzať vzniku chýb a ako zabezpečiť úspešnosť softvérového projektu. Na záver článok diskutuje súčasný stav v zodpovednosti za chyby a škody spôsobené chybami softvéru. Uvádza argumenty za aj proti zavedeniu zodpovednosti za chyby.

Úvod

Tvorba softvérových produktov je oblasť, v ktorej sa chyby vyskytujú v každej fáze vývoja projektu. Preto je veľmi dôležité brať do úvahy všetky možnosti vzniku chýb a čo najviac sa im snažiť vyvarovať. Chyby môžu nastať a vznikajú aj za ideálnych podmienok, ak je však manažment projektu zanedbaný, veľa vzniknutých chýb môže ostať neodhalených a šanca, že vzniknú nové chyby sa zväčšuje. Ak sa na zásadnú chybu narazí až v neskoršej fáze vývoja projektu, veľmi často sa stáva, že pokus o jej odstránenie hrubo naruší celý vývoj a môže projekt odsúdiť k zániku.

Preto je veľmi dôležité zaoberať sa pri manažmente softvérového projektu otázkou minimalizácie chýb v projekte na strane tvorby samotného kódu, ale rovnako aj na strane manažmentu softvérového projektu. V ďalšom texte podrobnejšie predstavíme oblasti, v ktorých často vznikajú chyby. Následne popíšeme postupy softvérového inžinierstva, vďaka ktorým môžeme vznik chýb znížiť a zároveň zabezpečiť skoršie odhalenie vzniknutých chýb. V poslednej časti sa zaoberáme zodpovednosťou za škody spôsobené chybami v softvéri a uvádzame argumenty za aj proti zavedeniu takejto zodpovednosti.

Manažment v softvérovom inžinierstve, december 2005, s. 1-6.

Časté oblasti vzniku chýb

Vznik chýb v softvérových projektoch je hlavne vďaka veľmi krátkej, iba niekoľko desaťročnej histórií softvérového inžinierstva bežná vec. Podľa prieskumu z roku 2003 [3] je priemerný počet chýb v softvéri na 1000 riadkov rovný 0.15, pričom najviac chýb softvér vykazuje, ak pochádza zo Spojených štátov amerických (až 0.4) a najmenej, ak pochádza z Japonska (0.02). Tieto čísla, pri zohľadnení faktu, že každý väčší produkt obsahuje aj viac ako milión riadkov kódu znamenajú, že počet chýb sa môže vyšplhať až k stovkám chýb v jednom produkte.

Ale aj napriek týmto nepriaznivým zisteniam sa čoraz viac darí pomocou zavádzania nových postupov odhaľovať vzniknuté chyby. Darí sa to najmä vďaka testovacím nástrojom ako je JUnit, statickou analýzou kódu, či používaním typovo bezpečných jazykov ako C# alebo Java. Čo sa nedarí znižovať je počet chýb, ktoré vznikajú pri tvorbe kódu, a to rovnako u začínajúcich programátorov, ako aj u skúsených programátorov s dlhoročnou praxou [4]. Zaujímavý je poznatok, že počet chýb u začínajúcich programátorov je približne rovnaký ako u tých skúsenejších, čo autori štúdie [4] odôvodňujú tým, že skúsenejší programátori pracujú na zložitejších úlohách a robia aj zložitejšie a ťažšie odhaliteľné chyby.

Veľkým problémom pri testovaní kódu môžu byť chyby v samotných testoch. Takéto chyby sa obyčajne rýchlo prejavia pri pokusoch testovať správny kód. Ak sa ale chyba v testovaní prejaví tak, že chybný test skončí úspešne pre správny aj chybný testovaný kód, môže to spôsobiť značné nepríjemnosti pri neskoršom testovaní produktu. Ďalším veľkým zdrojom nových chýb býva samotné odstraňovanie chýb. Na zamedzenie zavádzania nových chýb pri oprave starých chýb sa odporúča dôkladné testovanie všetkých dotknutých funkcií a modulov softvéru.

Manažment kvality softvéru sa okrem zabezpečenia zhody výsledného softvéru so špecifikáciou zaoberá aj kvalitou s ohľadom na počet chýb v softvéri. Z tohto druhého pohľadu je základným problémom kvality softvéru to, že programátori sú ľudia a ako takí aj oni robia chyby. Množstvo zanesených chýb do softvéru je ovplyvnené najmä zložitosťou a komplexnosťou systému. Čím je systém zložitejší, tým sa stáva menej prehľadný. Veľa skrytých chýb zostáva v softvéri aj po testovaní a navonok sa môžu prejaviť až pri dlhodobom nasadení, kedy môžu spôsobiť veľké škody. Chyby vznikajú aj počas prvotných fáz vývoja produktu, napríklad nepochopením požiadaviek zákazníka alebo zlým návrhom, ktorý je v rozpore so špecifikáciou. Tieto chyby stoja za mnohými predčasnými ukončeniami projektov a ich odstránenie je kľúčové pre úspešné dokončenie projektu. Úspešne vytvorený produkt zväčša obsahuje už iba chyby zanesené programátormi pri implementácií kódu. Väčšinou ide o preklepy, chyby z nepozornosti a chyby, ktoré vzniknú zanedbaním ošetrenia chybových stavov.

Bezpečnostné slabiny rozšírení softvéru

Bezpečnostné slabiny často vznikajú vďaka nedostatočne zabezpečeným rozšíreniam softvéru, ako je napríklad systém pluginov alebo podpora skriptovania v rámci softvéru. Záškodnícky kód, ktorý využíva takýto model, býva ťažko rozlíšiteľný od užitočného kódu a môže nastať situácia, kedy takéto rozšírenie prináša viac škody ako úžitku. Postupov ako sa chrániť pred záškodníckym kódom interpretovaným v rámci softvéru je viacero: analýza, prepísanie, monitorovanie a ďalšie. Analýza prebieha pred spustením kódu a v prípade zistenia nebezpečnosti je kód zamietnutý. Prepísanie je spôsob, pri ktorom modifikujeme nebezpečné časti kódu tak, aby nebol kód škodlivý. Monitorovanie prebieha počas vykonávania kódu a v prípade zistenia škodlivého kódu nastáva jeho zastavenie.

Možnosti zlepšenia kvality softvéru

V článku [1] autor uvádza množstvo poznatkov, ktorých aplikovaním môžeme skvalitniť proces vývoja softvéru. Ak budeme aplikovať tieto pravidlá, je pravdepodobné, že sa nám okrem skvalitnenia procesu vývoja podarí zároveň znížiť počet chýb v softvéri. Tu je niekoľko zásad pre vývoj softvéru, nielen z článku [1]:

Testovacie podmienky

Hneď ako je možné uzavrieť špecifikáciu podmienok, zmluvné strany by mali uvažovať ako bude možné otestovať splnenie požiadaviek. V skutočnosti mali nad týmto uvažovať už vo fáze tvorby špecifikácie, pretože je ľahké skĺznuť k testovacím podmienkam, ktoré sa nedajú reálne otestovať. Napríklad: „Program by mal uspokojivo spracovať vstup na výstup.“

Ak sa nám podarí správne špecifikovať podmienky pre testovanie, možnosť že chybu v programe neodhalíme je podstatne nižšia. Rovnako môžeme zabrániť vzniku chýb pri návrhu a implementácií vyplývajúcich z nepochopenia požiadaviek zákazníka.

Podrobná špecifikácia rozhraní

Presný popis jednotlivých formátov, špeciálne požiadavky na komunikáciu, ako napríklad kedy je možné prenášať niektoré typy dát. Všetky tieto detaily musia byť dohodnuté všetkými zúčastnenými stranami a spísané na papier. Táto podmienka v podstate patrí k tej predchádzajúcej. Ak má byť zabezpečená široká kompatibilita aj komunikácia pomocou implementovaných rozhraní, musí byť testovaná.

Skoré plánovanie a špecifikácia

Ak manažment pristúpi k tvorbe softvéru bez hlbšej analýzy a podrobného plánu, výsledkom bude odkladanie vytvorenia kompletnej špecifikácie a zdesenie, keď nevytvorenie jedného programu oneskorí vývoj celého systému. Zákonite potom vzniká časový stres a počet chýb vytváraných vystresovanými programátormi sa

zvyšuje. Čo môže mať za následok ďalšie oneskorenia pri vývoji alebo spôsobiť, že dodaný softvér bude nekvalitný a plný chýb.

Množstvo zamestnancov v počiatočných fázach

Analytici by nemali byť zaťažovaní povinnosťou zamestnať každého zamestnanca len preto, aby mal čo robiť. Kvantita nie je náhrada za kvalitu, môže veci iba zhoršiť. Je nutné, aby na návrhu systému pracovalo čo najmenej ľudí. O to viac by títo ľudia mali komunikovať medzi sebou, ale aj so zákazníkom.

Zodpovedná voľba jazyka

Výber programovacieho jazyka je rovnako ako výber celoživotného partnera ťažkou a dôležitou voľbou, ktorá sa nedá tak ľahko vziať späť. Ak už je programovací jazyk vybraný, stáva sa jeho zmena problematickou a projekt by to mohlo predražiť alebo úplne potopiť.

Plánovanie integrácie a nasadenia

Častá chyba pri plánovaní nasadenia softvéru do prevádzky je podcenenie času potrebného na integráciu softvéru po tom čo bol naprogramovaný. Pri integrácií vznikajú problémy aj pri veciach, kde by to nikto nečakal. Málo času vyhradeného na integráciu potom spôsobuje ďalší stres a prehliadanie menších nedostatkov na úkor zásadných chýb a problémov.

Skoré plánovanie testov

Akceptačné testy by mali byť určené čo najskôr, tak aby bolo možné testovať okamžite po implementácií. Mrhanie času pri vývoji softvéru sa nevypláca. Obyčajne sa totiž v takýchto prípadoch začnú objavovať problémy vtedy, keď už ich nikto nečakal a nie je na ne čas.

Testom riadené programovanie

Testy píšeme pred samotným písaním implementácie vlastností. Cyklus písania testov a písania funkčného kódu sa opakuje v malých iteráciách a často. Každá funkcia musí mať test ešte pred napísaním jej kódu. Testom je pokrytý každý riadok kódu programu. Funkčný testovací kód predstavuje okamžitú odozvu pri návrhu nových funkcií. Každý píše vlastné testy pre vlastný kód. Vývojové prostredie musí zabezpečiť rýchlu kompiláciu a služby pre testovanie. Návrh musí pozostávať z vysoko súdržných, voľne previazaných komponentov tak, aby bolo testovanie jednoduché.

Zodpovednosť za chyby

Prístup softvérových spoločností k zodpovednosti za škody spôsobené chybami softvéru je v súčasnosti žalostný. Ak si v dnešnej dobe inštalujete ľubovoľný softvér,

musíte väčšinou odsúhlasiť text zmluvy, v ktorej sa autor zbavuje akejkoľvek zodpovednosti za chyby v softvéri. Takéto zbavenie zodpovednosti je logické a akceptovateľné ešte v prípade, že softvér je používateľovi ponúkaný zadarmo, bez nároku na odmenu, ak je však softvér poskytnutý za úplatu, je takéto správanie v slušnej spoločnosti nanajvýš podozrivé. Tieto praktiky sú dnes široko používané a väčšina používateľov ich nakoniec akceptuje, ale takéto správanie je v iných oblastiach podnikania neprípustné. Tak prečo by práve oblasť softvérového inžinierstva mala byť výnimkou? Ukazuje sa, že aj napriek textom zmlúv, ktoré musia používatelia akceptovať pri inštalácií, niektoré súdy majú na zodpovednosť za škody spôsobené chybami v softvéri iný názor a prisudzujú postihnutým používateľom odškodné [2]. Aj z tohto dôvodu sa softvérové spoločnosti začínajú viac venovať dôkladnému testovaniu vlastného softvéru pred uvoľnením na trh. Podobné zmluvy však určite v dohľadnej dobe nezmiznú. Ide hlavne o problém previazanosti rôznych softvérových výrobkov medzi sebou a fakt, že takéto zmluvy používajú súčasné operačné systémy, a preto softvér, ktorý sa vykonáva na týchto operačných systémoch nemôže vziať na seba zodpovednosť, ktorej sa zbavili tvorcovia operačných systémov.

Argumenty tvorcov softvéru

Tvorcovia softvéru sa bránia proti zavedeniu zodpovednosti za chyby a škody nimi spôsobené. Snažia sa presvedčiť verejnosť, že v súčasnosti nie je v ľudských silách zabezpečiť bezchybnosť softvéru, nie je možné úplne eliminovať chyby, že by tým mohlo dôjsť k spomaleniu vývoja softvéru, zvýšili by sa náklady na vývoj softvéru, a tým by došlo ku krachom viacerých spoločností, softvér by sa predražil. Často môže byť veľmi ťažké určiť pôvodcu chyby. Argumentujú, že o kvalitu softvéru sa postará samotné trhové prostredie a že súčasná podpora je dostatočná.

Argumenty používateľov

Medzi hlavné argumenty používateľov softvéru patrí skvalitnenie softvéru, čo je dôležitejšie ako rýchlosť vývoja nových verzií. Ďalej argumentujú stále vyššou mierou používania softvéru v bežnom živote ľudí a tým pádom stále väčšiu závislosť ľudí od softvéru. Softvér sa používa v podstate v každom odvetví ľudskej činnosti, od riadenia výroby v továrňach až po ovládanie rakiet s nukleárnymi hlavicami. Preto treba donútiť tvorcov softvéru k vyššej zodpovednosti, a tým zvýšiť kvalitu ich produkcie. Ďalším argumentom používateľov sú súčasné zákony pre ostatné oblasti ľudského obchodu. Zákazník je chránený obchodným zákonníkom 24 mesačnou zárukou, ktorá však na základe výnimky neplatí pri softvérových výrobkoch, pričom softvér je rovnaký produkt ako čokoľvek iné a nie je preto dôvod obrať zákazníka o jeho spotrebiteľské výhody.

Záver

Chyby sú súčasťou vývoja softvéru od vzniku softvérového inžinierstva a budú realitou aj v budúcnosti. Najväčší pokrok sa v súčasnosti deje v oblasti testovania kódu,

pomocou ktorého je možné odhaľovať viac chýb ako v minulosti. Ďalší pokrok je možný v oblasti špecifikácie a návrhu softvéru a v oblasti dodržiavania zásad vývoja bezpečného softvéru. Najväčšie rezervy stále ostávajú na poli zodpovednosti za chyby v softvéri, kde si tvorcovia softvéru stále nedokážu priznať vlastnú zodpovednosť.

Použitá literatúra

1. Barry W. Boehm: Software Engineering As It Is, *4th International Conference on Software Engineering*, September 1979, pp. 11-21.
2. Michael A. Cusumano: Who is liable for bugs and security flaws in software? *Communications of the ACM*, Vol. 47, No. 3 (March 2004), 25-27
3. Michael A. Cusumano et al.: Software development worldwide: The state of the practice. *IEEE Software*, (Nov.-Dec. 2003).
4. Gerard J. Holzmann: The Logic of Bugs. *ACM SIGSOFT Software Engineering Notes*, Vol. 27, No. 6 (November 2002), 81-87

Annotation

Errors and management of software project

Every software development process needs to count with the emergence of errors. The emergence of errors is inevitable in every human activity, therefore it is necessary to deal with this problem and try to remove the errors. In this document we talk about critical places of software, where errors are common. We talk about plugin and script systems and their impact on the control of error behaviour and the security of software. Next we describe options in management to prevent emerging of errors in software projects. In the end we discuss recent progress in responsibility for errors and damage caused by errors in software.