LOV CHÝB NIE JE JEDNODUCHÝ

Ak test nezdetekuje žiadnu chybu, nemusí to ešte značiť, že tam žiadna nie je. Chybný môže byť samotný test.

Ondrej Kallo

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
ondrej[zavináč]kallo[.]sk

Abstrakt. *Zákazníkovi pri kúpe softvérového produktu ide hlavne o to, aby tento fungoval správne a umožnil mu tak bez problémov vykonať všetko čo potrebuje. Softvér pred jeho vypustením firmy testujú, aby sa zabezpečila čo možno najväčšia správnosť. Ako je potom možné, že sa pri jeho používaní objavujú chyby? V tejto eseji uvádzam niekoľko príčin, ktoré často vedú k zlyhaniu celého procesu testovania. Obvykle ide hlavne o zanedbanie testovacej fázy pri vývoji či jej skrátenie kvôli blížiacemu sa termínu, ďalej príprava testov na poslednú chvíľu alebo zlý výber testovacích vstupov. Keďže jasné riešenia úplne všetkých problémov s prípravou a plánovaním testov neexistujú, uvádzam určité odporúčania, ktoré by mali pomôcť zefektívniť proces testovania.*

Kľúčové slová: *testovanie softvéru, príprava testov, plánovanie testov, problémy, odporúčania*

Úvod

Kvalita produktu je relatívny pojem. Každý z nás si pod ním zrejme predstaví niečo iné podľa toho, čo pre nás musí spĺňať daný produkt, aby sme ho považovali za naozaj kvalitný. Jedna z definícii vraví, že kvalita je súhrn vlastností a charakteristík produktu, ktoré preukazujú jeho schopnosť splniť určené alebo odvodené potreby. Ide teda o stupeň splnenia požiadaviek, ktoré sú kladené na daný produkt, a nie je to len nejaká absolútna hodnota [1].

Zákazník si v prvom rade vyberá softvér, ktorý mu pomôže s jeho úlohami, až potom zvažuje ostatné aspekty ako sú cena, dizajn a pod. Dôležitá je teda správnosť programu,

aby používateľovi naozaj uľahčil robotu a nenarobil mu viac škody než úžitku. Mieru správnosti softvéru obvykle preukazujeme testovaním.

Testovanie softvéru môžeme definovať ako vykonávanie daného programu na zistenie, či zodpovedá svojej špecifikácii a je vykonateľný v cieľovom prostredí [4]. Dôležité je uvedomiť si, že testovanie je dynamický proces, potrebujeme mať hotovú vykonateľnú jednotku. V prípade, že testovaním detekujeme chybu, je potrebné ju nájsť a odstrániť. Aj takýmto spôsobom prispievame k vyššej kvalite výsledného produktu.

Napriek všetkej snahe testérov sa však chyby bežne v softvéri vyskytujú. Testovanie ani statické metódy validácie síce nezaručujú, že sa podarí nájsť a opraviť všetky chyby, no podľa mojej mienky je výskyt niektorých chýb spôsobený aj nedokonalým procesom testovania. Rozhodol som sa preto preskúmať bežné chyby, ktorých sa ľudia vzhľadom na testovanie dopúšťajú, a ako sa im čo najlepšie vyhnúť.

Plánovanie je dôležité

Kedy je najvhodnejší čas začať s testovaním? Firmy zvyknú nechávať testovanie na koniec, tesne pred vypustením produktu na trh, aby „vychytali posledné muchy." V prípade, že termín sa nebezpečne približuje, je testovanie prvým kandidátom, kde sa šetrí časom. Takéto konanie mi ale pripadá ako hazardovanie, pretože sa šetrí, kde by sa nemalo.

Z vlastnej skúsenosti totiž viem, že tento prístup s veľkou pravdpodobnosťou vedie ku katastrofe. Pri tvorbe jedného zadania som odložil všetky testy až na čas, keď bude úplne všetko naimplementované. Keď mi potom testovanie začalo ukazovať chyby, zistil som, že jediný elegantný spôsob na ich odstránenie by bola radikálna zmena architektúry programu. Na to mi však neostal vôbec čas, a tak som sa musel uspokojiť s pridaním záplat, ktoré ošetrovali detekované chybné stavy. Výsledkom však bol veľmi neefektívny program, ktorého zdrojový kód bol rozsiahlejší, než by bolo treba. A aby toho nebolo dosť, tak výsledok bol potom tak nečitateľný, že aj ja ako autor som sa v ňom prakticky nevyznal, a to už tesne po napísaní!

Na mojej skúsenosti vidieť potvrdenie pravidla, že čím neskôr sa chyba objaví, tým viac prostriedkov je potrebné vynaložiť na jej opravu. A keď sa všetko robí na poslednú chíľu, často je už na reálnu nápravu neskoro. Testovanie je preto potrebné plánovať, vyčleniť naň čas, aby sa naozaj stihlo vykonať. Vidím to však tak, že dôležitejšie je dbať na to, aby sa testovalo priebežne už počas implementácie. Nielenže tak objavíme chyby skôr, ale v menších celkoch sa potom ľahšie aj hľadá zdroj chyby.

Fáza príprav testov

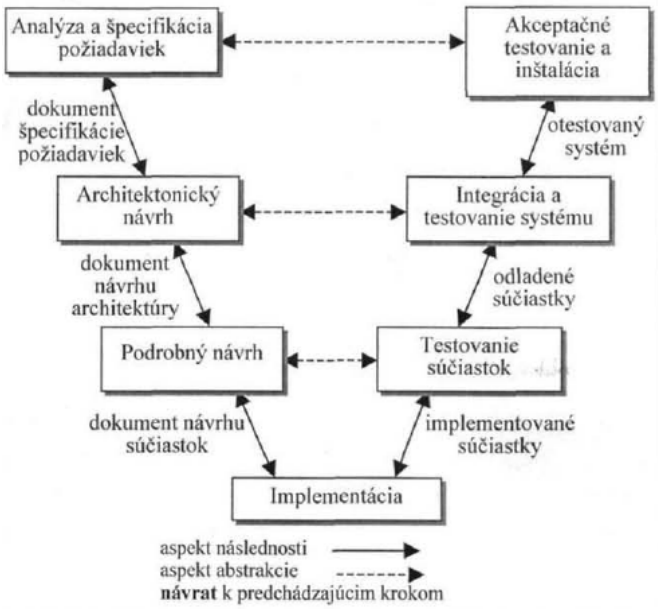
Ak chceme testovať softvér, musíme mať vopred pripravené testovacie prípady. Tieto zahŕňajú zvolené vstupy a k nim príslušné výstupy, ako aj postupnosť akcii, ktorými by sme sa mali k daným výstupom dopracovať. Kedy je však správna doba na prípravu?

Ako som už spomenul v úvode, nutnou podmienkou testovania je existencia vykonateľnej jednotky. Mohli by sme si povedať, že vytvoríme takúto jednotku, potom pripravíme testy a následne ich aplikujeme. Pri tomto postupe sa takmer určite stane, že zabudneme otestovať nejakú časť funkcionality danej jednotky, a to tým pravdepodobnejšie, čím väčšiu časť vytváraného produktu testujeme. Ak totiž čakáme

s prípravou na dokončenie celého systému, ťažko si už budeme pamätať na presnú špecifikáciu jednotlivých súčiastok, voči ktorej by prebiehala ich validácia.

Jedno z riešení by som videl v príprave návrhu jednotlivých testovacích prípadov ešte pred implementáciou, už počas rôznych etáp návrhu. Výborne to podľa mňa vystihuje V-model vývoja softvéru (pozri *Obr. 1*), ktorý zachytáva súvis jednotlivých etáp vo vývoji nielen z aspektu následnosti, ale aj z aspektu abstrakcie [1].

Fáza analýzy a špecifikácie požiadaviek je priam ideálna na prípravu testovacích prípadov pre akceptačné testovanie, pretože tu sme presne na úrovni zákazníkových požiadaviek, voči ktorým akceptačné testovanie prebieha. Vo fáze architektonického návrhu zase môžeme navrhnúť testovacie prípady, ktoré umožnia otestovať úspešnosť integrácie súčiastok do podsystémov a systém ako celok. Umožňuje nám to úroveň abstrakcie, na ktorej sa pohybujeme. Pri zjemňovaní návrhu na úroveň jednotlivých súčiastok zase navrhneme, ako budeme implementované súčiastky odlaďovať.



Obr. 1. V-model vývoja softvéru.

Konkrétne testy pripravíme potom na základe hotových návrhov. Je výhodnejšie pripravovať testy pred alebo až po implementácii danej časti, ktorá má byť testovaná? Podľa McConnella [3] je niekoľko dôvodov, ktoré hovoria pre prvú možnosť:

- Či pripravujeme testy pred alebo po písaní kódu, trvá to vždy rovnako dlho.
- Ak máme testy pripravené vopred, chybu sa podarí skôr odhaliť, a teda ľahšie odstrániť.
- Písanie testov pred samotnou implementáciou núti vývojárov viac sa zamyslieť nad špecifikáciou a návrhom, čo vedie k lepšiemu kódu.
- Na chyby v špecifikácii sa príde skôr, pretože je veľmi obtiažne chystať testy pre zlú špecifikáciu.

Vytváranie testov pred samotnou implementáciou je základom jednej z techník používaných v rámci extrémneho programovania. Ide o vývoj riadený testovaním, kedy tvorba programu prebieha po malých prírastkoch. Na začiatku sa vytvorí množina testov, potom sa implementuje daná funkcionalita a spustia sa testy. Ak tieto nie sú úspešné, prebehnú úpravy naprogramovanej časti. Takto sa to opakuje až pokým testy úspešne nezbehnú. Vtedy sa pridajú nové testy a celý cyklus začne odznova. Toto pokračuje kým nie je vývoj ukončený [2].

Problémy s výberom testovacích vstupov

Najväčším problémom pri príprave testov je zrejme výber vhodných vstupných parametrov a poradia, v akom ich zadávame. Zrejme jediná možnosť, ako detekovať všetky chyby, by bola vyskúšať všetky možné kombinácie vstupov. Už pre jednoduché úlohy je však toto množstvo priveľké, pre reálne systémy sú to už nepredstaviteľné čísla. V skutočnosti sa teda vyberá len malá podmnožina vstupov, ktoré sa potom použijú. Tento výber bohužiaľ nemá stanovené žiadne presné pravidlá, ktoré by nám jasne určili vstupy spôsobujúce problémy, preto uvediem určité odporúčania, podľa ktorých je dobré sa riadiť.

Pri výbere sa treba snažiť pokryť čo najrôznejšie možné vstupy. Ak totiž spustíme test s nejakými vstupmi a žiadna chyba sa nenájde, je vysoko pravdepodobné, že pre veľmi podobné vstupy bude výsledok rovnaký a jediným výsledkom bude zbytočná strata času. Rovnako je zbytočné používať dve sady vstupov, ak pomocou nich detekujeme presne tie isté chyby.

Jedným z najčastejšie používaných kritérií pri výbere testovacích vstupov býva testovanie hraničných hodnôt. Ak máme určenú maximálnu alebo minimálnu hodnotu niektorej premennej, skúsime ju postupne nastaviť menšiu, väčšiu a rovnú danému obmedzeniu. Hodnoty medzi takýmito hranicami pritom môžeme považovať za rovnako dobré bez ohľadu na to, ktorú nakoniec zvolíme. Komplikovanejšie to začne byť v prípade, že potrebujeme určiť hodnoty viacerých premenných, ktoré sú spracovávané súčasne a môžu sa navzájom ovplyvňovať. Vtedy už musíme brať do úvahy, aký môže byť výsledok ich vzájomnej interakcie [4].

Testovacie vstupy by mali čo najlepšie pokrývať vytváraný program. Testéri sa snažia, aby sa im podarilo vybrať také vstupy, aby sa každý riadok v kóde, každý príkaz vykonal aspoň raz. Toto kritérium je síce celkom dobré, aspoň pre začiatok, ale rozhodne nie je postačujúce. Oveľa lepšie je, ak zvládnu pokryť všetky vetvy v programe. V reálnych systémoch toto naozaj nie je ľahké dosiahnuť. Ak nám však stačí vykonať každý príkaz aspoň raz, je potrebné si zistiť, koľko rôznych ciest vedie cez program a spraviť toľko testov, koľko treba, aby všetky tieto cesty pokryli. Steve McConnel vo svojej knihe Code Complete ponúka jednoduchý algoritmus, vďaka ktorému si vieme vypočítať minimálny potrebný počet takýchto testov [3]:

1. začneme číslom 1 za prvý priamy prechod programom
2. pripočítame 1 za každé z nasledujúcich kľúčových slov alebo ich ekvivalentov: *if*, *while*, *repeat*, *for*, *and* a *or*
3. pripočítame 1 za každú vetvu v *case* výraze; ak *case* výraz nemá vetvu *default*, pripočítame ešte raz 1

Výsledkom algoritmu je minimálny počet testov, ktorými je možné pokryť všetky cesty v programe. Nie je to však možné dosiahnuť akýmikoľvek testami – tieto treba starostlivo zostaviť. Kľúčom sú tie slová, pri ktorých sme pripočítavali 1. Tie reprezentujú výrazy, ktoré môžu nadobúdať hodnoty pravda (*true*) alebo nepravda (*false*). Správnym priradením hodnôt premenným, ktoré sa v nich vyskytujú, získame hľadané testovacie prípady.

Každý možný vstup patrí do práve jednej tzv. triedy ekvivalencie. Ak sa pri použití daného vstupu zistí chyba, rovnakú chybu detekujeme pri použití ľubovoľnej inej hodnoty z danej tried. Toto nám dáva určitú voľnosť, že si môžeme vybrať, ktorú hodnotu v rámci niektorej triedy použijeme. Výhodné je preto voliť si také vstupné hodnoty, ktoré nám umožnia ľahko overiť ručným výpočtom, či je výsledok vrátený programom správny.

Automatické testovanie

Automatizácia už od priemyselnej revolúcie uľahčuje ľuďom prácu, a tak niet divu, že je snaha automatizovať aj testovanie. Pri vývoji riadenom testovaním je potrebné vykonať veľké množstvo testov. Kým pre človeka je to nesmierna záťaž, počítač to zvládne ľahko a rýchlo. Automatické testovanie taktiež nájde uplatnenie pri regresnom testovaní, kedy pomocou starých testov zisťujeme, či nám čerstvo pridaná nová, resp. pozmenená stará funkcionalita nevniesla chybu do iných už otestovaných častí programu. Ak by mal človek znova a znova vykonávať tie isté úkony, viedlo by ho to k istej apatii a ľahko by prehliadol prípadnú chybu. Ďalšou výhodou automatizácie voči ľuďmi vykonávanému testovaniu je, že nám umožňuje jednoduchšie vykonávať záťažové testy. Môžeme nasimulovať situácie, kedy je do systému prihlásených veľa používateľov a systém teda musí vykonávať veľa akcií naraz.

Tu by som však chcel varovať všetkých, ktorí automatické testovanie považujú za všeliek na neduhy testovania. Iste, je to veľmi užitočná pomoc, no aj tu sa môžu ukrývať problémy, ktoré netreba podceniť.

V prvom rade ide o to, že testy navrhujú ľudia, takže sa ľahko môže stať, že testovanie nebude pokrývať všetky prípady, ktoré by malo. Ďalším problémom je, že tieto testy sú tiež len počítačové programy, a teda zrejme budú obsahovať chyby. Ak test nenájde žiadnu chybu, ešte to nemusí znamenať, že tam žiadna nie je. Možno je chyba v samotnom teste. Môže sa taktiež stať, že test je falošne pozitívny, čo má za následok stratu času hľadaním chyby, ktorá reálne vôbec neexistuje.

Psychologické aspekty testovania

Pri testovaní vstupuje je dôležitým faktorom aj ľudská psychika. Naše podvedomie nás dokáže ovládať častokrát viac, než sme ochotní si pripustiť.

Vidíme len to, čo chceme vidieť

Keď sa chystáme testovať program, je chybou, ak k tomu pristupujeme s postojom, že aj tak to určite funguje správne. Môže sa nám potom stať, že hoci test chybu odhalí, my ju prehliadneme. Ľudia totiž často veci vnímajú ako by ich chceli vidieť, resp. ako si myslia, že by to malo byť, a nie ako tomu v skutočnosti naozaj je. Príkladom je aj rébus na *Obr. 2*,

kde väčšina ľudí na prvý pohľad prečíta text „paris in the spring," ale vôbec si neuvedomia, že je tam jedno slovo „*the*" navyše.

Obr. 2. Nie každý si všimne zdvojeného slova the.

Určite má každý podobné skúsenosti ako ja, keď mi až niekto druhý objavil chybu v dokumentácii či zdrojovom kóde, pretože predtým som to ja čítal tak, ako som to chcel napísať a nie to, čo som naozaj napísal. Je preto nanajvýš potrebné pristupovať k testovaniu so snahou chybu objaviť.

Vytváraný produkt netreba šetriť

Vývojári, ktorí pracujú na nejakom programe, zvyknú potom testovať, či všetko funguje. To znie z ich pohľadu celkom logicky, avšak cieľom testovania má byť hľadanie chyby. Na testovanie sa teda môžeme pozerať aj ako na deštrukčnú činnosť, ktorej podstatou je snaha zisťovať, či existuje spôsob, ako program zhodiť. Ak však testovanie vykonáva sám autor, môže sa stať, že svoje dielo podvedome šetrí, a tak mu môžu niektoré chyby uniknúť.

Pri väčších projektoch je preto výhodné využívať služby testovacích tímov. Títo testéri jednak nie sú zaujatí, a navyše ich skúsenosti z predchádzajúcich projektov im umožnia prísť na viac trikov, aké by inému nemuseli ani prísť na um. Pri menších projektoch, kedy zostavenie špeciálneho tímu nemá zmysel, by sa malo na testovaní podieľať viacero vývojárov, pričom každý by prispel svojou troškou nápadov, čo by mohlo viesť k chybe. Ak človek pracuje na projekte sám, nezostáva mu nič iné, len sa snažiť odosobniť sa.

Záver

Je známy Dijkstrov výrok „Testovanie nemôže preukázať, že v programe nie sú chyby. Môže iba ukázať, že tam chyby sú." Aj najdokonalejšie testovanie v súčasnosti deteguje priemerne len okolo 60% chýb [3]. V kombinácii so statickými metódami validácie môžno toto percento výrazne zvýšiť, ale ani tak nemožno dokázať, že program naozaj funguje so 100%-nou správnosťou.

Ak pochybíme v pocese testovania, znížime už aj tak nie celkom postačujúcu schopnosť detekcie chýb. V tejto eseji som sa snažil poukázať na faktory, ktoré znižujú kvalitu testovania a tým pádom zvyšujú množstvo chýb, ktoré má šancu uniknúť v rámci finálneho produktu. Zároveň som sa snažil prispieť aj riešeniami alebo odporučeniami, ako eliminovať tieto negatívne javy.

Použitá literatúra

1. Bieliková, M.: *Softvérové inžinierstvo – princípy a manažment.* Vydavateľstvo STU, Bratislava, 2000.
2. George, B., Williams, L.: *An Initial Investigation of Test Driven Development in Industry.* In: Proc. of the 2003 ACM Symposium on Applied Computing, 2003.
3. McConnell, S.: *Code Complete, Second Edition*. Microsoft Press, Redmond, 2004.
4. Whittaker, J. A.: *What Is Software Testing? And Why Is It So Hard?* In: IEEE software, vol. 17, no. 1 (January 2000)

Annotation

The hunt for bugs is not easy

When buying a software product, the customer is mainly concerned about its correctness so that it will help him to perform all tasks he need to be done without any problems. Companies perform software testing before its release so that the maximum possible correctness is achieved. How is it possible that there are always some defects left that emerge only after the product is being used by customers? In this essay, I introduce several possibilities that may often lead to malfunction of whole testing process. It usually happens when the testing phase is neglected or shortened because of approaching deadline, when test cases are prepared at the last possible moment or because of wrong selection of test inputs. Not all problems with test preparation and planning have clear solution so I give only some basic advices that should help to make testing process more effective.