Testovanie ako základný kameň zabezpečenia kvality

BC. PAVOL DANIŠ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
pavol[.]danis[zavináč]centrum[.]sk

Abstrakt. Manažment kvality by mal byť pri výrobe alebo vývoji akýchkoľvek produktov stavaný do popredia. V rámci softvérového inžinierstva obzvlášť. Našim cieľom je predsa spokojný zákazník, pričom spokojnosť je do značnej miery ovplyvnená kvalitou softvéru. V eseji rozoberám, aké metódy sa používajú v oblasti manažmentu kvality, pričom sa zameriavam na zrejme najznámejšiu metódu – testovanie. Z množstva literatúry a teda aj množstva názorov na testovanie chcem vybrať pre čitateľa podľa môjho názoru najlepšie techniky ako, kedy a čo testovať. K slovu sa dostane aj metodika vývoja softvéru založená na testovaní - Test driven development(TDD). Poukážem na hlavné výhody tejto modernej metodiky používanej v moderných prístupoch k vývoju softvéru – agilné a extrémne programovanie.

Úvod

Zabezpečenie kvality je v procese vývoja softvéru dôležitým faktorom. Kvalita sa všeobecne pri softvérových produktoch určuje odlišne ako u bežných produktov. Špecifickosť spočíva najmä v rôznorodosti používateľov. Jeden si nevie produkt vynachváliť, pre iného je ťažkopádny a možno až nepoužiteľný.

Z predchádzajúceho vyplýva, že zabezpečenie kvality rozhodne nie je triviálna záležitosť. O to viac sa jej treba však venovať, ak chceme, aby bol koncový používateľ spokojný. Kvalitu produktu z majoritnej časti ovplyvňuje kvalita procesu vývoja produktu. Dalo by sa povedať, že tieto dva druhy kvality sú priamo úmerné. Existujú však aj rôzne metódy, ktoré majú zaistiť kvalitu. Iné metódy slúžia na odhalenie nekvalitných častí produktu.

V nasledujúcich častiach uvediem niektoré metódy, ktoré zabezpečujú kvalitu, pričom neskôr sa zameriam na testovanie a TDD. Chcem poukázať na výhody moderných prístupov k vývoju softvéru v rámci zabezpečenia kvality. Ako teda testovať, aby sme dosiahli vyššiu kvalitu softvéru, vyvíjaného agilnými metodikami?

Manažment projektov softvérových a informačných systémov, október 2008, s. 1-8.

Metódy zaisťujúce kvalitu

Profesionálny a fundovaný softvérový inžinier môže byť podľa mňa iba ten, ktorý vytvára kvalitný softvér. Tí, ktorý sa chcú stať profesionálmi v tomto obore, by tiež mali dbať na kvalitu softvéru. Dosiahnuť to môžu najmä ak sa budú riadiť odporúčaniami uznávaných expertov a využívať nimi zavedené a osvedčené metódy zabezpečujúce kvalitu.

Z metód a techník, ktoré môžeme v našej snahe za kvalitným produktom využiť, spomeniem nasledujúce[2, 4]:

- testovanie
- štruktúrované prechádzanie kódu
- revízia návrhov
- refaktorizácia
- McCabova cyklomatická metrika zložitosti (meria zložitosť rozhodovacích štruktúr)
- zložitosť dedenia
- množina závislostí
- pomery prepojenia tried

Testovanie

Hádam najznámejšou metódou na zabezpečenie kvality je testovanie. Vyplýva to z logiky procesu tvorby produktov – aj iných ako softvérových. Každý výrobca by si mal otestovať svoj produkt predtým, ako umožní používať tento produkt zákazníkovi. V opačnom prípade produkt otestuje až zákazník, čo ho môže stáť pri zistení chýb dobré meno, zákazníkov a z toho plynúci finančný prospech.

Preto by sme nemali fázu testovania podceňovať. Prakticky každá metodika vývoja softvéru kladie silný dôraz na testovanie. V jednej z prvých a aj najznámejších modelov životného cyklu – vodopádovom modeli – sa testovanie považuje za samostatnú fázu vývoja nasledujúcu po implementácii. V nových metodikách – agilnom aj extrémnom programovaní je testovanie považované za základný kameň kvality vyvíjaného softvéru a prípravám testov sa venuje pozornosť už pred implementáciou[5].

Môže sa stať, že sme pri vývoji mali pocit, že sme venovali dostatočnú pozornosť testovaniu, no nakoniec sme aj tak neodhalili niektoré chyby. Príčinou môže byť, že si nie sme istý *čo, ako a kedy* treba testovať.

Čo testovať

Ak máme existujúci zdrojový kód, vyvstáva otázka, čo z tohto kódu má byť testované. Výborné by bolo, ak by testy pokryli všetky riadky kódu. Ideálne by však bolo, ak by testovanie pokrylo všetky stavy, v ktorých sa produkt môže nachádzať.

Predpokladajme napríklad, že máme funkciu, ktorá spracováva dve celé čísla, pričom každé z nich môže nadobúdať hodnotu 0 až 999.

```
int test(int a, inb) {
    return a / (a + b);
}
```

Táto troj riadková funkcia má teoreticky 1 000 000 logických stavov, pričom 999 999 bude pracovať korektne a jeden nesprávne – keď a + b = 0. Tu vidíme, že aj pri otestovaní každého riadku nemusíme odhaliť každú chybu. Otestovať však každý stav v bežných projektoch je zrejme málo reálne, keď len tento malý kus zdrojového kódu má toľko stavov[2].

Predpokladajme teda, že je nereálne pri väčších projektoch v rozumnom, niekedy ani v reálnom čase, prejsť všetky stavy programu. Riešením je prejsť pri testovaní každý riadok kódu a správne zvoliť testovaciu množinu dát. Testovacie dáta by mali pokrývať hraničné i bežné stavy. Tieto dáta by mali byť aj reálne aj syntetické. Reálne pochádzajú z reálneho prostredia používania, predstavujú typické používateľské dáta. Syntetické sú umelo vytvorené – kvôli veľkosti množiny dát, dáta pre otestovanie hraničných podmienok a podobne[2].

Zdá sa to ako ťažká úloha, zabezpečiť, aby testovanie pokrylo všetky riadky kódu. Nasledujúce typy testovania sú v tomto veľmi nápomocné a využívané v moderných prístupoch k vývoju softvéru[2]:

- testovanie jednotky (unit testing)
- testovanie celku
- validácia a overovanie
- testovanie výkonu
- testovanie použiteľnosti

Testovanie jednotky je najdôležitejšou časťou testovania. Ak totiž nefunguje samostatná jednotka, nemôžeme predpokladať, že v celku bude táto jednotka funkčná. Pred zapojením do celku by mala byť teda overená funkčnosť samotných modulov. Až po tejto fáze môžeme previesť testovanie celku.

Toto testovanie má preveriť spoluprácu subsystémov projektu. Má sa overiť, či podľa analýz a návrhov správne medzi sebou komunikujú jednotlivé moduly a či v rámci celého systému plnia svoje poslanie a povinnosti voči ostatným modulom.

Validácia a overovanie má zistiť, či požiadavky na funkčnosť softvéru sú v zhode s používateľskými potrebami.

Testovanie výkonu overuje, ako sa softvér správa pri záťaži a v extrémnych podmienkach. Nesmieme zabudnúť, v akých podmienkach bude používateľ softvér

používať – počet používateľov, pripojení, uskutočnených transakcií, používaný hardvér a podobne.

Testovanie použiteľnosti je iný typ, aké som spomínal doteraz. Ide o testovanie v reálnych podmienkach prevádzky softvéru. Testovanie uskutočňuje reálny používateľ. Softvér by mal byť pre používateľa ľahko použiteľný. Sleduje sa, ktoré činnosti sú vykonávané najčastejšie, či nevzniklo nedorozumenie pri požiadavkách a podobne. Zozbierané informácie využijeme pre úpravu častí softvéru. Z tohto dôvodu je dobré, aby sa táto fáza vykonala dosť skoro. Pri väčších projektoch sa odporúča aj zapojenie odborníkov na ľudský faktor.

Kedy a ako testovať

Na otázku kedy testovať je pomerne zložité. Každá metodika vývoja softvéru či model životného cyklu má testovaciu fázu v stanovenom časovom horizonte, závisí teda na konkrétnom tíme, akú metodiku používa a aké zvyklosti daný tím má. Netreba byť však lenivý a nechať si testovanie na poslednú chvíľu.

Práve naopak. Testovanie a vytváranie testov by malo prísť čo najskôr. Pri vzniknutom kuse zdrojového kódu, po vytvorení modulu, alebo dokonca pred začatím písania zamýšľaného kódu –TDD.

Väčšina testov by mala prebiehať automaticky a aj výsledky by mali byť interpretované automaticky. Spúšťať testy by sme mali čo najčastejšie, samozrejme záleží od potrebného času na test. Pri testoch, ktoré vyžadujú viac času, môžeme porozmýšľať o nočnom testovaní, prípadne testovanie raz do týždňa alebo mesiaca. Testovanie by každopádne malo byť pravidelné. Treba zvážiť časovú náročnosť testov a tvorby zdrojového kódu, prípadne ďalšie špecifické požiadavky daného projektu a vytvoriť plán testovania, ktorý bude dodržiavaný.

Test driven development

Aktuálne veľmi populárne a často rozoberané metodiky – agilné programovanie a extrémne programovanie kladú na testovanie veľký dôraz. Metodika TDD vznikla na základe vytvárania testov pred vytvorením zdrojového kódu.

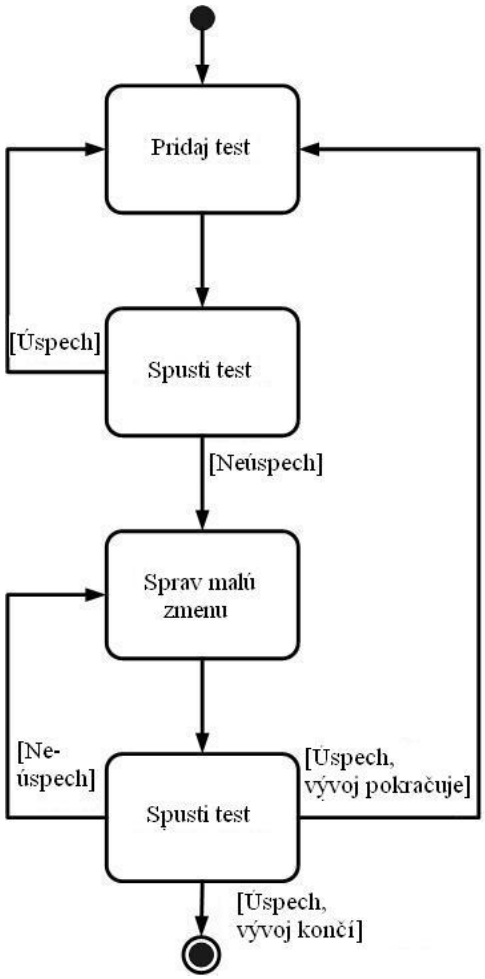
Táto praktika sa môže zdať čudná. Mnohí programátori nemajú radi ani samotné testovanie a nevenujú sa mu tak ako by sa patrilo. Teraz sa od programátora dokonca vyžaduje, aby písal testovanie ešte predtým, ako napíše kód.

Pri striktnom dodržaní tohto pravidla však získame páku na zabezpečenie kvality. Vďaka tomuto prístupu totiž môžeme prísť pri správnom napísaní testov k výsledku, kedy budeme mať každý riadok otestovaný. A tu už sme veľmi blízko k určeniu kvality. Vieme totižto ako sa správa celý kód, i keď stále nemusíme vedieť ako sa správa program v každom logickom stave, do ktorého sa môže dostať.

Ako teda postupujeme pri metodike TDD? Mali by sme sa riadiť nasledovnými krokmi:

1. určíme si novú funkcionalitu softvéru, ideálne malého rozsahu
2. pripravíme test pre túto funkcionalitu
3. spustíme test, ktorý má zlyhať
4. napíšeme zdrojový kód riešiaci žiadanú funkcionalitu
5. opäť spúšťame test
6. ak test zlyhal, opravíme kód a prejdeme na krok 5
7. ak test prešiel úspešne, môžeme prejsť na krok 1 alebo ukončiť vývoj softvéru

Na obrázku 1 je postupnosť krokov vývoja podľa TDD znázornená diagramom aktivít v UML notácii. Odporúčam si pozrieť aj zdroj [3]. Tu môžeme vidieť podľa autora ideálne časové úseky na jednotlivé fázy vytvárania kódu v rámci TDD. Vývoj na základe TDD metodiky by sme mali rozumieť v súvislosti s agilným prístupom, kedy skracujeme jednotlivé iterácie do čo najmenších časových úsekov. Častá by mala byť aj refaktorizácia kódu[5].



Obr. 1. Diagram aktivít postupu pri vytváraní testov v TDD[1].

Ak chce tím vyvíjať softvérový produkt na základe tejto metodiky, programátori by mali byť pripravený na to, čo ich čaká. Ideálne by bolo, aby sa rýchlo naučili disciplíne písania testov a nevynechať prípravu ani jedného testu. Ani keď sa zdá byť daná časť triviálna. Murphyho zákony platia samozrejme aj v softvérovom inžinierstve. Chyba vyvstáva aj v častiach, v ktorých to vôbec nečakáme. Preto nestačí dúfať či veriť, že je všetko v poriadku. Treba to overiť.

Pre zachovanie disciplíny by bolo ideálne párové programovanie[5], ktoré sa vyskytuje v extrémnom programovaní. Jednotlivec si sám pred sebou ľahšie zdôvodní, prečo niečo neurobiť, aj keď by to mal urobiť. Druhý z páru dohliadne na prvého, aby

si plnil svoje predsavzatia povinnosti. Samozrejme takouto praktikou nie sú nadšený zamestnávatelia, keďže sa im zdá, že týmto prístupom sa im zvyšujú náklady. Často si však neuvedomujú, ako sa takýmto prístupom minimalizujú náklady pri údržbe softvéru.

Záver

Zabezpečenie kvality je široká téma. Ja som sa snažil poukázať na zrejme najznámejšiu a dôležitú časť zabezpečenia kvality – testovaniu. A to najmä z pohľadu nových prístupov k vývoju softvéru.

Opisujem, čo treba testovať, ako a kedy to testovať. Snažím sa poukázať na to, že netreba podceniť testovanie v rámci vývoja softvérového produktu a venovať sa mu čo najskôr. Moderné metodiky vývoja softvéru kladú veľký dôraz na testovanie. Opisujem aj metodiku založenú na samotnom testovaní a príprave testov – Test Driven Development.

Podľa mňa je dôležité, aby testovanie prebiehalo pokiaľ možno automaticky, pravidelne, aby sme testovali každý modul, ako aj celý softvérový produkt a aby sme testovali často a včas, samozrejme na správne zvolených testovacích dátach. Ak bude spĺňať testovanie tieto atribúty a vývoj softvéru bude vedený kvalitným procesom, mala by sa úroveň kvality softvéru výrazne zvýšiť.

Použitá literatúra

1. Ambler S.W.: Agile database techniques, Ambysoft Inc., Dostupné na http://www.agiledata.org/essays/tdd.html (17.10.2008)
2. Hunt A., Thomas D.: Programátor pragmatik, Computer Press, a.s., 2007. Vydanie prvé
3. Kadlec V.: Agilné programovanie, Computer Press, a.s., 2004.
4. Meijer L: Test-driven development, Unit Test, VSTS, NUnit, TestDriven.NET, whats all this?, Dostupné na http://leonmeijer.nl/archive/2007/06/05/45.aspx (17.10.2008)
5. Wells D.: Extreme programming, Dostupné na http://www.extremeprogramming.org/ (17.10.2008)

Annotation

Testing as the corner-stone of quality assurance

Management of quality is very important in the development of any product. In the software engineering especially. Satisfied customer is our goal and satisfaction is largely influenced by quality of the software. The essay discusses the methods used in the field of quality management, focusing on perhaps the most popular method - testing. From the lot of literature and lot of opinions about testing I want to choose the best practices of what, how and when to

test software products. Methodic of developing software based on testing – Test Driven development will also be analyzed. I will try to point out the advantages of this modern methodic used in modern approach of developing software – agile and extreme programming.