Veľké testovanie v malom

JÁN HLAVÁČ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
janohead@yahoo.com

Abstrakt. Proces vývoja softvérového produktu sa skladá z niekoľkých základných častí, okrem analýzy, návrhu a implementácie riešenia nesmieme zabudnúť na testovanie a s ním spojené zabezpečenie kvality. Testovanie získava, čím ďalej, tým väčšiu váhu, vyžaduje si stále viac a viac prostriedkov i času, a to sú len niektoré dôvody, prečo je potrebné sa venovať metódam aplikovania a plánovania testovania. Táto práca sa snaží ukázať dôležitosť testovania a vysvetliť jeho základné princípy. Základom je opis celkového prístupu k testovaniu vo väčšom projekte, ktorý postupuje podľa klasického V-modelu s využitím systému 4CC a porovnanie s ďalšími typmi modelov. Opis prechádza od metodiky a modelu vývoja, cez zloženie tímu a jednotlivé funkcie až po dokumentáciu k testovaniu.

Zámerom je ukázať možnosti využitia opisovaného prístupu k testovaniu veľkého projektu na malom projekte. Pomocou analogického postupu v riadení testovania a upravením funkcií členov projektového tímu dostávame model, ktorý by mal pomôcť pri realizácii testovania v menšom tíme.

Úvod

Smerovanie vývoja softvéru je ovplyvňované mnohými vplyvmi, rovnako ako vývoj v ostatných častiach hospodárstva. Na softvér sa od začiatku pozeralo ako na produkt, teda ako na niečo, čo sa bude predávať a uspokojovať požiadavky zákazníkov. Spočiatku to vyzeralo, ako keby bol „pánom“ softvér a zákazníci len čakali, čo im prinesie a čo bude vlastne tento softvér robiť. Samozrejme obraz sa radikálne zmenil a dnes softvér, lepšie povedané softvérové spoločnosti, doslova a do písmena slúžia a pracujú podľa požiadaviek zákazníkov.

Manažment v softvérovom inžinierstve, október 2006, s. 1-9.

Softvérový priemysel je veľký obchod s nesmiernou konkurenciou a rýchlym napredovaním. Práve konkurencia a snaha presadiť sa je hlavným motívom softvérových spoločností zlepšovať seba a svoje produkty.

Produkty sú úspešné, ak sú kvalitné, hlavne lepšie ako konkurenčné, a preto sa dnes vynakladá obrovské úsilie na zabezpečovanie kvality produktov. Jednou z hlavných častí zabezpečenia kvality je testovanie produktov, ktorému sa budeme ďalej venovať.

Pohľady na testovanie

O testovaní sa popísalo mnoho, metodiky, odporúčania, odborné práce. Názory boli a aj budú ešte dlhú dobu rôzne na to ako testovať, kedy testovať, kto by mal testovať a dokonca aj čo to znamená testovať. Niekto hovorí, že testovanie je umenie, iní, že je to veda. Mne sa zdá akoby sa testovanie zmenilo z umenia, kedy išlo naozaj o talent testera, na vedu, ktorej sa samostatne venuje mnoho odborníkov. Avšak cieľom tejto práce nie je polemizovať nad významom pojmu testovanie, a preto budeme vychádzať zo všeobecnej definície [2] „Softvérové testovanie je proces používaný na identifikáciu správnosti, úplnosti a kvality vyvíjaného počítačového softvéru. V skutočnosti, testovanie nikdy nevie zaručiť správnosť počítačového softvéru, čo môže byť spravené len formálnou špecifikáciou."

Minulosť ukázala testovanie ako veľmi dôležitú časť v životnom cykle vývoja softvéru, teraz si to plne uvedomujeme. Aj dnes sa však niekedy na testovanie zabúda alebo sa k nemu pristupuje s menšou vážnosťou. Tento nedostatok je akoby v podvedomí ľudí. Najčastejšie sa s ním stretávame v neskúsenom, prípadne menšom kolektíve, ako napr. študentské tímové projekty.

Príčin je viacero, naozajstná neskúsenosť a podcenenie testovania alebo všetkým známy časový faktor. Nedostatku času a pretečeniu termínov sa mnoho krát nevyhne ani veľký projektom. V tomto prípade sa problém rieši upravením rozsahu testov, ba v kritických prípadoch i vypustením niektorých testov, čo v praxi znamená podstupovanie obrovského rizika.

Účinnou formou prevencie proti zanedbaniu testovania je plánovanie testovania a kontroly kvality. Plánovanie je nezávislé od veľkosti projektu a malo by sa robiť stále. Rozdiel je samozrejme v dostupných zdrojoch ako technických, finančných tak aj ľudských. Plánovanie testovania a testovanie sú neoddeliteľnou súčasťou projektu a zaberajú značnú časť jeho zdrojov, ale zároveň ich aj šetria. Prečo? Odpoveď je jednoduchá, testovanie a odhalenie chyby včas je rádovo lacnejšie ako jej neskoršia oprava.

Základné princípy

Cieľom testovania je odhaliť chyby produktu, zlepšiť jeho kvalitu a spoľahlivosť. Každý produkt má vlastnú špecifikáciu, požiadavky a proces vývoja, a teda je zrejmá

potreba pripraviť vhodné testovanie. Testovanie závisí od už spomínaných požiadaviek, zvoleného vývojového procesu a dostupných zdrojov.

Ako testovať?

Testujeme tak, aby sme pokryli čo najväčšiu množinu čo najdôležitejších prípadov použitia vyvíjaného produktu. Z toho vyplýva potreba usporiadania jednotlivých prípadov testovania podľa dôležitosti, inak priorita testovania [5].

Napríklad, ak vytvárame systém pre obsluhu zdravotníckeho zariadenia, potrebuje v prvom rade testovať jeho bezpečnosť vzhľadom na pacientov a personál (safety), ak vytvárame zabezpečovací systém, tak je pre nás kritická jeho odolnosť a stabilita (security).

Pri regresnom testovaní alebo pri záťažových testoch, teda všade tam, kde sa opakujú činnosti sa snažíme zaviesť v čo najväčšej miere automatizáciu testovania, ktorá nám ušetrí čas a ľudské zdroje [4].

Model testovania by mal byť prispôsobený modelu vývoja produktu, aby bola zabezpečená synchronizácia testovania s vývojom a konzistencia samotného testovania.

Kedy testovať?

Tu postupujeme podľa jednoduchého pravidla. Čas na testovanie nasleduje po implementácii. Uvedené pravidlo je síce jednoduché, ale jeho dôsledky sú hlbšie, ako sa na prvý pohľad zdá.

Základ je testovanie jednotlivých častí systému, modulov, hneď po ich implementáci. Ak vytváraný systém obsahuje viacero modulov je potrebné testovať ich jednotlivo, aj ako jeden celok a vzájomnú interakciu, ide o integračné testovanie. Postupnosť testov by mala byť zvolená tak, aby sme dokázali identifikovať, ktorá časť systému vyvolala prípadnú zmenu.

Dá sa predpokladať, že sa budú jednotlivé moduly meniť a opravovať, a preto aj opakovane testovať. Tu však nestačí len otestovanie príslušného modulu, ale celej časti systému, s ktorou interaguje, a tak by sme mali po každej zmene robiť okrem testovania modulov a integračného testovania aj regresné (systémové) testovanie. Inak povedané testovanie striktne plánované na základe V-modelu nie je správne a potrebuje zabezpečiť určité opakovanie testovania v iteráciach (pozri Obr. 1).

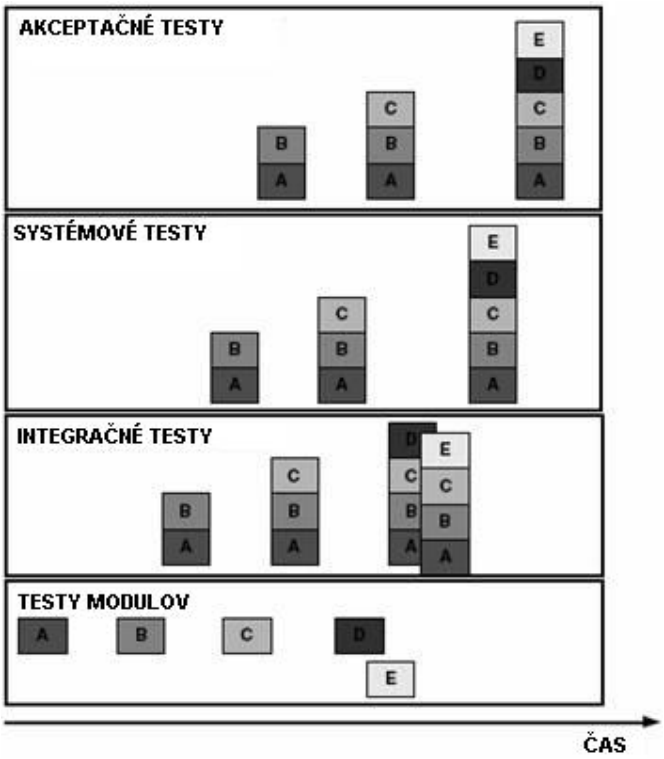
Čo testovať?

V ideálnom prípade testujeme všetky prípady použitia a všetky konfigurácie systému a vstupov, ale tento prípad je ojedinelý z dôvodu nedostatku zdrojov („všetky“ v zmysle všetky triedy vstupov, konfigurácií ... nie v zmysle celej množiny vstupov). Keďže sme ohraničený našimi možnosťami je potrebné si definovať množinu kritických testov podľa vybraného kritéria, ako to bolo uvedené v časti „Ako testovať?“.

Kritické testy sú testy, ktoré musí produkt nevyhnutne spĺňať vyplývajúc zo špecifikácie. Túto množinu postupne rozširujeme o ďalšie testy najlepšie prioritne usporiadané.

V podstate sa testovaním sledujú nasledujúce vlastnosti softvérového produktu:

1. funkcionalita
2. spoľahlivosť
3. použiteľnosť
4. efektívnosť
5. udržovateľnosť
6. prenositeľnosť



Obr.1: Časové usporiadanie testovania

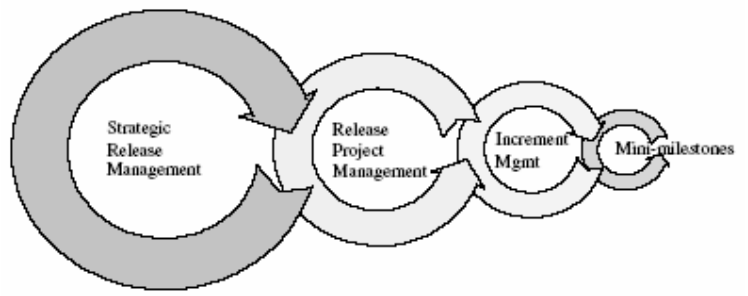
Manažment testovania

Žiadny projekt sa nezaobíde bez plánovania a manažmentu, záleží na ich kvalite aký kvalitný bude výsledok projektu a ako efektívne sa využijú dostupné zdroje. K dispozícii máme niekoľko metodík a modelov. Sem patrí aj systém 4CC (Four

Cycles of Control), ktorý odstraňuje nedostatky najznámejšieho a stále používaného V-modelu.

Systém 4CC [3] definuje štyri časové rámce alebo cykly, ktoré majú ozrejmiť vzťahy medzi testovaním a ostatnými aktivitami projektu, taktiež definujú základné otázky, na ktoré je potrebné odpovedať v danom čase (pozri Obr. 2).

- *Strategic Release Management* - zaoberá sa stratégiami uvoľňovania riešení a slúži ako rozhranie medzi manažmentom a vývojármi, pri riešení základných otázok ohľadne návrhu a vývoja.
- *Release Project Management* - rieši otázky týkajúce sa konkrétneho projektu so zameraním na uvoľnenie konečného riešenia.
- *Increment Management* manažuje - jednotlivé úpravy v návrhu, urdžuje čiastkové výsledky a zodpovedajúcu dokumentáciu.
- *Mini-milestones* - slúžia na štruktúrovanie a plánovanie každodennej práce so zameraním na jednotlivé role v rámci projektu.



Obr.2: Cykly systému 4CC [3]

Okrem V-modelu a jeho rôznych variant poznáme i niekoľko ďalších modelov pre manažment testovania, ako príklad uvedieme *Test-driven development (TDD).* Testami riadený vývoj TDD [1] je založený na princípe najprv test potom zdrojový kód. Je to diametrálne odlišný prístup ako v prípade V-modelu, základom je definovanie testov pred implementáciou funkcií a okamžitou aplikáciu testov na systém. Výsledkom je, že testy skončia neúspechom a to je impulz na začatie implementácie, ale len po bod kedy testy prebehnú úspešne. Nasleduje refaktorizácia a úprava resp. pridanie nových testov, ktoré opäť končia neúspechom, a následná implementácia pridanej funkcionality. V tomto prípade testová dokumentácia, testy ako také, plnia i funkciu špecifikácie, pretože definujú požiadavky na systém, pre ktoré test prebehne úspešne.

Tím testerov a ako to funguje

Tím špecializovaný na testovanie je veľmi vhodným riešením testovania v rámci projektu. Podmienkou je dostatok ľudí pracujúcich na projekte, a preto sa využíva

hlavne vo veľkých spoločnostiach. Za ideálny pomer programátor : tester sa vraj považuje pomer 1:1, ale to je v skutočnosti len veľmi ťažké dosiahnuť.

Výhodou testovacích tímov je ich nezávislosť a špecializácia, tým sa odbremeňujú hlavne programátori. Testeri sa priamo nepodieľajú na riešení, a tak pri testovaní môžu využiť iný pohľad na problém ako by mal programátor. Vo všeobecnosti by ten istý človek nemal testovať vec, ktorú sám vyvinul.

Mal som možnosť podieľať sa skoro na „vzorovom“ projekte väčšieho rozsahu. Testovací tím bol oddelený od vývojových tímov a plnil aj funkciu komunikácie so zákazníkom (Client Team ďalej CT). CT bol rozdelený na menšie skupiny (tiež tímy) podľa jednotlivých častí vyvíjaného systému. Nech má systém tri hlavné časti tak, potom aj CT sa skladá z troch tímov. Štruktúra CT kopíruje štruktúru vývojových tímov, a teda istému počtu programátorov zodpovedá určitý počet testerov.

Projektová hierarchia a úlohy

Hlavou celého projektu je ako vždy *project manager*, ktorý koordinuje projekt, tímy a na najvyššej úrovni rieši problémy so zákazníkom na základe podkladov získaných od tímov.

Team manager sa stará o chod svojho tímu (CT, ...) a koordináciu s ostatnými tímami, ako napr., kedy bude release, code freeze perióda, alebo pre nás zaujímavé, kedy a čo sa bude testovať.

Team lead vedie svoj tím (subteam), riadi a rozdeľuje konkrétne činnosti a pracuje spolu s členmi. Pri testovaní to znamená konzultovanie s manažérmi a ostatnými team_leadmi, príprava prípadov testovania a testovacích skriptov, príprava hlásení a výsledkov testovania (testovacej dokumentácie) a samotné testovanie.

Člen tímu, *tester*, samozrejme testuje a zaznamenáva výsledky testov, v prípade neúspešného testu zadefinuje incident, ktorý rieši zodpovedný programátor. Riešenie incidentu si často vyžaduje konzultáciu testera a programátora. Úlohou testera je okrem testovania aj príprava ďalšieho testovania, teda definícia testovacích prípadov a tvorba k nim prislúchajúcich testovacích skriptov. Tester si najprv musí naštudovať riešenie a jeho funkcionalitu, aby mohol zvoliť správne testovacie metódy.

Jednou z najdôležitejších úloh CT je akceptačné testovanie s klientom. Pri teste je prítomný klient, tester a podľa potreby aj niekto z manažmentu. Tester v tomto prípade pripravuje testovanie a odpovedá na otázky kladené klientom, keďže on zodpovedá za testovanie danej funkcionality.

Testovacie prostredia

Testovanie ako aj implementácia je časovo náročná činnosť. Sériovým vykonávaním implementácie a testovania strácame zbytočne veľa času. Vhodným plánovaním dokážeme tento čas značne zredukovať. Ak testujeme paralelne s implementáciou, samozrejme testujeme n-tú iteráciu a implmentujeme n+1-vú iteráciu, nemusíme čakať na dokončenie testov pre predošlú iteráciu.

Paralelné testovanie je možné len ak máme k dispozícii viac systémových prostredí. Jedno prostredie určené na testovanie, jedno na implementáciu, jedno reálne

produkčné prostredie, ak systém už beží a podobne ... Počet prostredí závisí od požiadaviek a možností projektu.

Nevýhodou viacerých prostredí je potreba ich udržiavať synchronizované a konzistentné, teda ak sa niečo zmení na jednom prostredí musí byť zadefinovaný proces ako túto zmenu preniesť aj na ostatné prostredia, ale to je už problém pre manažment prostredí.

Dokumentácia testovania

Testovanie je zložitý proces skladajúci sa z množstva podprocesov a cyklov, ktorých výsledky potrebujeme vyhodnotiť a uchovať pre ďalšie časti projektu. Projektový manažér a všetci členovia manažmentu plánovania potrebujú poznať výsledky testov aby dokázali identifikovať stav projektu a naplánovať nasledujúce akcie. Bez výsledkov testov sa nezaobídu ani programátori, pretože až po úspešnom ukončení testov sa môžu aplikovať nové časti systému. A samozrejme i samotní testeri potrebujú zaznamenávať stav vykonávaných testov.

Toto sú hlavné dôvody, prečo je nevyhnutné vytvárať a udržovať testovaciu dokumentáciu, ktorá slúži na riadenie, prezentáciu výsledkov a vizualizáciu procesu testovania. Dokumentácia by mala obsahovať aspoň nasledujúce dokumenty:

- *Test Workplan And Test Cycles* – dokument obsahuje zoznam testovacích cyklov spolu s dotknutými časťami systému a plán testov, ktoré sa budú vykonávať v rámci daných testovacích cyklov. Plán je tvorený pre stredne dlhé obdobie a môže kopírovať cykly pre dodávanie systému (napr. mesiac, 14 dní ...)
- *Test Case* – opisuje testovaný prípad, teda aká funkcionalita ktorej časti systému je testovaná, aké sú podmienky pre testovanie a očakávaný výsledok. Obsahuje odkaz na testovacie skripty, ktoré testujú daný prípad.
- *Test Script* – je to presne definovaná postupnosť krokov na vykovanie konkrétneho testu. Obsahuje identifikačné údaje skriptu (identifikátor, testovací prípad, testovací cyklus), popis činnosti, prípadne potrebné informácie k prostrediu alebo aplikácii, ktorá sa má vyžiť. Vo všeobecnosti platí, že podľa testovacieho skriptu by mal byť schopný vykonať opisovaný test aj nezainteresovaný človek, ako napríklad klient.
- *Traceability Matrix* – slúži ako tabuľka, pomocou ktorej sme schopní sledovať celú štruktúru testovania. Mala by obsahovať všetky testované časti systému, testované funkcionality, priradenie testovacích prípadov a testovacích skriptov. Účelom dokumentu je sledovanie testov v rámci testovacieho procesu.
- *Test Statu Reort* – správa o stave testovania je najčastejšie žiadaným dokumentom zo strany manažmentu testovania, keďže obsahuje aktuálny stav každého z testov. Záznam o teste obsahuje jeho identifikátor a zaradenie v rámci testovacej hierarchie, dátum vykonania a stav, ktorý nadobúda jednu so štyroch hodnôt, ako napr. naplánovaný, vykonávaný, úspešný alebo neúspešný.

Spomenutá štruktúra nie je nijako záväzná a samozrejme sa prispôsobuje podľa potrieb projektu.

Testovanie v malom

Doteraz sme opisovali testovanie na veľkom projekte, ktorý mal k dispozícii samostatný testovací tím. Dajú sa aplikovať spomínané metódy na malý projektový tím o veľkosti päť až šesť ľudí?

Je jasné, že malý projekt má značné obmedzenia na strane zdrojov, a to hlavne ľudských, na druhej strane odpadá zložitá réžia a manažment veľkej skupiny osôb. Hlavným problémom je rozdelenie úloh v rámci tímu, ako manažment plánovania, manažment kvality, vývoj. Existuje jediné praktické riešenie, ktoré spočíva v zlučovaní resp. prekrývaní úloh. Predpokladá sa, že každý člen malého tímu bude zároveň vyvíjať produkt a okrem toho plniť aj nejakú zo spomínaných funkcií.

Teda o celý proces testovania by sa mal starať jeden člen tímu, ktorý bude podporovaný ostatnými členmi prevažne vo fáze vykonávania testov. Jeho úlohami budú plánovanie testovania, príprava a vykonávanie testov, a tvorba testovacej dokumentácie.

Ak sa pokúsime o náznak mapovania úloh medzi veľkým a malým tímom dostaneme nasledovné možnosti:

- testovací tím – osoba zodpovedná za kvalitu a testovanie, podpora ďalších členov v prípade potreby
- testovacia dokumentácia – všetky časti dokumentácie sú aplikovateľné v rovnakej miere, dokumentácia nezávisí od veľkosti tímu
- testovacie prostredia – testovacie prostredia je vhodné využívať aj v tomto prípade, i keď výsledné zlepšenie nemusí byť až také zreteľné. Rozdiel bude v počte a prevedení týchto prostredí, môžu sa využiť rôzne verzie toho istého systému s cieľom simulovať skutočný paralelizmus.

Metodika a modely vývoja softvéru, ktoré zahŕňajú aj manažment testovania, uvedené v dokumente sú definované univerzálne a ľahko prispôsobiteľné potrebám konkrétneho projektu.

Záver

Testovanie je nevyhnutné pri každom vývoji softvéru, či už v malom alebo vo veľkom tíme. Pri jeho aplikácii nám pomáhajú známe modely a ich zlepšenia prostredníctvom systémov pravidiel. Vhodnou voľbou modelu dosiahneme zreteľné zefektívnenie procesu plánovania a celkového manažmentu testovania, nesmieme však zabudnúť na tvorbu testovacej dokumentácie, ktorá v niektorých prípadoch dokáže určovať smerovanie celého projektu. Výsledkom správneho testovania je zvyšovanie kvality produktov a úspechu na trhu.

Použitá literatúra

1. Ambler, S.W.: Introduction to Test Driven Development (TDD). 2006, [cit 2006-10-26].
   URL: < http://www.agiledata.org/essays/tdd.html >.
2. Dictionary LaborLaw Talk: Software testing, [cit 2006-10-26].
   URL:< http://dictionary.laborlawtalk.com/Software_testing>.
3. Pyhäjärvi M., Rautiainen K., Itkonen J.: Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects. *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2002.
4. Rees, E., Fryer, L.: IGDA Business Committee - Best Practices in Quality Assurance/Testing. 2003.
5. Voas, J., Miller, K.: Semantic Metrics for Software Testability. *The Journal of Systems and Software, Elsevier Science Publishers Ltd,* 1993.

Annotation

Small big testing

Process of software development consists of several basic parts, like analysis, design and implementation, but we shouldn't forget testing and quality assurance (QA). Testing becomes very important in the process and consumes the main part of system resources and time. These are only few reasons why should we concentrate on software testing management. This essay deals with basics rules of software testing and describes 4CC framework for improving the standard V-model. In the next part we describe realization of testing process in large project team later on, we discuss the possibilities of using described process in a small project team of five or six members.