I NAPRIEK KVALITE VŽDY TESTUJ

Chyby sa nájdu všade, aj vo mne.

Maroš Bednár

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
Autor[zavináč]mail[.]com

Abstrakt. *Rozmanitosť a zložitosť softvéru vzrastá zo dňa na deň, preto musí byť zabezpečená kvalita softvéru. Kvalita softvérového systému je posudzovaná viacerými atribútmi kvality, či už je to výkon a použiteľnosť pre koncového používateľa alebo vysoká stabilita a opätovné použitie. Kladenie dôrazu na kvalitu softvéru zlepšuje vývoj systému a predchádza sa ním nejasnostiam pri samotnej implementácií. Testovanie je oproti zabezpečeniu kvality softvéru orientované na "detekciu". Pri testovaní kvality softvéru sa odhaľujú nedostatky a chyby vyvíjaného softvéru, preto je nevyhnutné testovať softvér, aj keď sa zdá, že nie je potrebné. V súčasnosti existuje mnoho rôznych metód ako správne testovať kvalitu softvéru. Esej sa sústreďuje na zabezpečenie kvality softvéru a presvedčí Vás, že vývoj riadený testami šetrí drahocenný čas a finančné prostriedky s porovnaním testovania softvéru po dokončení implementácie.*

Kľúčové slová: *zabezpečenie kvality v softvérovom systéme, testovanie softvéru, vývoj riadený testami*

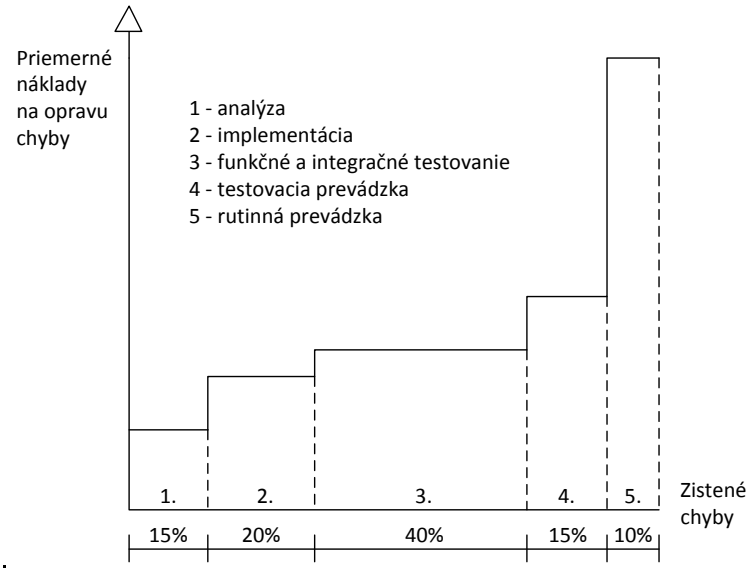
Úvod

Prudký rozvoj informačných technológií prináša neustále nové požiadavky na vývoj softvéru. Softvérové spoločnosti sú nútené rýchlo reagovať na nové IT trendy, ak chcú obstáť v náročnej konkurencii.

Od informačných technológií sa vyžaduje vysoká spoľahlivosť, pretože výpadok systému alebo zásadná programová chyba môže spôsobiť značné škody. Zákazník vždy požaduje systém, ktorý spĺňa jeho požiadavky a vyskytuje sa v ňom čo najmenej chýb,

najlepšie žiadna. Kvalita systému nezáleží len od celkového času stráveného na vývoji systému až po zavedenia do prevádzky. Systém sa môže vytvárať akokoľvek dlho, pokiaľ nebol dostatočne kvalitne vyvíjaný a chyby sa začnú prejavovať po samotnom nasadení softvéru, tak cena rapídne stúpa opravou vzniknutých chýb (Obr. 1.). Zistenie chyby v neskoršom štádiu znamená návrat k predošlým fázam vývojového cyklu. Pre tvorbu softvéru platia priam požiarnické zásady. Čím skôr sa oheň nájde a hasí, tým menšie škody vzniknú.

Cieľom tejto eseje je oboznámenie sa s najrozšírenejšími modelmi vývoja softvéru a detailnejší pohľad na vývoj riadený testami.



Obr. 1. Obvyklé štatistické rozdelenie detegovaných chýb a nákladnosť [6].

Kvalita softvérového systému

Nároky na kvalitu sa oproti minulosti značne zmenili. Vtedy sa softvér prispôsoboval skôr hardvéru, keďže cena softvéru bola omnoho menšia ako hardvér. Teraz je to presne naopak, hardvér sa prispôsobuje softvéru. Keďže každý človek si môže vysvetliť pojem kvalita softvéru akokoľvek, tak existuje norma ISO/IEC 9126 [2], ktorá vo svojej prvej časti hovorí, že softvér je kvalitný, ak obsahuje atribúty ako funkčnosť, spoľahlivosť, použiteľnosť, efektívnosť, udržateľnosť a prenosnosť (Tab. 1.).

V prípade väčšiny softvérových systémov, dosiahnutie cieľov kvality softvéru z veľkej časti závisí od dostupnosti zdrojov pre vykonanie aktivít potrebných pre zlepšenie kvality softvéru. Medzi všeobecné pravidlo patrí, čím viac je dostupných prostriedkov na vývoj a testovanie, tým bude softvér kvalitnejší.

Na vývoj softvéru spĺňajúceho jednotlivé atribúty kvality je potrebné stanoviť čo najdôkladnejšiu analýzu spoločne so špecifikáciou softvéru. Podľa mojich doteraz získaných skúseností som zistil, že zle definovaná špecifikácia znamená mnohé chyby v neskorší etapách vývoja softvéru. Niekedy malá zmena v systéme môže spôsobiť, že je potrebné zmeniť značnú časť počiatočnej špecifikácie. Ideálny plán vývoja softvéru je

vtedy, keď zákazník stanoví celkovú funkcionalitu systému. Avšak tento ideálny plán sa zrealizuje málokedy, pretože vývoj softvéru je ovplyvňovaný viacerými faktormi. Najčastejšie to bývajú zmeny funkcionality spôsobené zákazníkom. Pri dopĺňaní rôznych častí systému, ktoré neboli špecifikované na začiatku vývoja systému, vznikajú často chyby. Preto je nevyhnutné testovanie systému po dokončení aj akejkoľvek malej časti systému aby bola dodržaná kvalita softvéru.

Tab. 1. Klasifikácia vlastností a atribúty kvality softvéru podľa normy ISO.

Vlastnosť	**Atribút**	**Vlastnosť**	**Atribút**
Efektívnosť	Časová zložitosť	Prenosnosť	Hardvérová nezávislosť
	Využitie zdrojov		Softvérová nezávislosť
Funkčnosť	Úplnosť		Inštalovateľnosť
	Správnosť		Znovu použiteľnosť
	Bezpečnosť	Spoľahlivosť	Dokonalosť
	Kompatibilita		Tolerancia chýb
	Interoperabilita		Dostupnosť
Udržiavateľnosť	Opraviteľnosť	Použiteľnosť	Zrozumiteľnosť
	Rozšíriteľnosť		Naučiteľnosť
	Testovateľnosť		Prevádzkyschopnosť
			Spoľahlivosť

Rational Unified Process

Metodika Rational Unified Process (RUP) sa stále viac rozširuje vo firmách zaoberajúcich sa vývojom softvéru. Táto metodika patrí primárne k tradičným metodikám. Opakom sú potom agilné prístupy na vývoj softvéru.

Táto metodika bola vyvinutá spoločnosťou Rational Software Corporation, ktorá ju i dnes zdokonaľuje. Metodika RUP vychádza z kolekcie osvedčených praktík a postupov pri vývoji softvéru. Je založená na objektovom prístupe. V súčasnosti existujú silné prepojenia a väzby na rôzne CASE nástroje. Je použiteľná pre akýkoľvek rozsah projektu, no vďaka vysokej rozsiahlosti je vhodné prispôsobiť metodiku špecifickým potrebám. RUP je vhodnejší skôr pre rozsiahlejšie projekty a väčšie vývojárske tímy, keďže kladie dôraz na analýzu a dizajn, plánovanie, riadenie zdrojov a dokumentáciu.

Tak ako všetky metódy, žiadna nie je bez chyby. Pri tejto vidím veľkú nevýhodu v tom, že ak má RUP obsahovať všetky možné prípady vo vývoji softvéru (čo môže byť veľký problém), tak popisuje aj veľa krokov, ktoré sú vo väčšine projektov zbytočné a tým sa projekt predlžuje.

Microsoft Solutions Framework

MSF je všeobecná metodika pre riadenie procesov pri tvorbe softvérových systémov založených na skúsenostiach rôznych softvérových firmách. Na rozdiel od iných popisných metodík sa však každá konkrétna implementácia MSF už nepozerá na vývoj softvéru a jeho životný cyklus z hľadiska vonkajšieho pozorovateľa, ale z aktívneho hľadiska účastníka všetkých procesov, krokov ich činností, aktivít vedúcich k finálnemu produktu.

MSF i konkrétne metodiky dodávané spoločnosťou Microsoft a ďalší partneri umožňujú organizáciám tvoriť si vlastné implementácie alebo upravovať existujúce metodiky. Všeobecná metodika MSF voľne definuje tri hlavné oblasti, ktoré potom konkrétna metodika použije pri tvorbe softvéru a môže ich svojím špecifickým spôsobom napĺňať, a to:

- Základný princíp: vyjadruje hodnoty a štandardy, ktoré sú spoločné všetkým ostatným atribútom rámca. Tieto princípy sú spoločné pre všetky roly účastníkov softvérového projektu. Spoločne vyjadrujú a formulujú základné, jednotné a zrozumiteľné prístupy, ako riadiť tímy a procesy.
- Tímový model: popisuje prístup ako organizovať pracovníkov a ich aktivít s cieľom vytvoriť a dodať úspešný produkt.
- Model procesu: ide o iteratívny špirálový model obsahujúci viacnásobné série krátkych vývojových cyklov nadväzujúcich na seba.

MSF ponúka viacero výhod. Oproti ostaným metodikám MSF je flexibilná, rozšíriteľná, je možné ju upraviť a prispôsobiť tak, aby vyhovovala potrebám každého veľkého a zložitého projektu. Patrí medzi najčastejšie používané metodiky, ktoré sú integrované do existujúcich prostredí.

Extrémne programovanie

Extrémne programovanie (XP) patrí medzi najnovšie a najrozšírenejšie agilné metódy vývoja softvéru. Ide o tradičné činnosti, ktoré sú ale dopracované do extrémov. Vďaka tomu by malo byť extrémne programovanie schopné sa lepšie prispôsobovať meniacim sa požiadavkám zákazníka a dodať softvér na vyššej kvalite.

XP je metodika pre malé až stredne veľké programátorské tímy (2-10 ľudí), ktoré vyvíjajú softvérové projekty zo zadaní, čo sa môžu často a rýchlo meniť. Najlepší efekt sa dosahuje predovšetkým na menších projektoch, kedy je viac-menej lepšia a častejšia komunikácia so zákazníkom ako pri rozsiahlejších projektoch, pričom netreba zabudnúť na ráznosť testovania. Zo skúseností môžem potvrdiť, že najviac chýb vzniká nedorozumením v komunikácií vývojárov.

Zapojením zákazníka do vývoja softvéru má svoje výhody aj nevýhody. Medzi hlavnú výhodu patrí rýchla odpoveď zákazníka na hotové časti funkcionality. Nevýhodou býva častá zmena požiadaviek na systém.

Testovanie

Akingbehin prehlásil [1] :

"Testovanie je aktivita, ktorou sa ustanovuje nevyhnutná dôvera, že program alebo systém robí to, čo má robiť, na základe súboru požiadaviek, ktoré má používateľ stanovené"

Na meranie kvality softvéru sa používa testovanie. Je to proces, pri ktorom sa odhaľujú chyby. Kedysi sa testoval len finálny produkt. Teraz je testovanie neoddeliteľnou súčasťou počas celého vývoja softvéru.

Testovanie negarantuje, že softvér je automaticky kvalitný a neobsahuje chyby. Kvalitu softvéru hodnotíme na základe očakávaných vlastností, ktoré sa testujú. Tieto vlastnosti sa delia do 3 skupín [5]:

- Vlastnosti vonkajšej kvality: spoľahlivosť, použiteľnosť
- Vlastnosti vnútornej kvality: efektívnosť, funkčnosť
- Vlastnosti neskoršej kvality: udržiavateľnosť, prenosnosť

Softvér je nevyhnutné testovať počas celého vývoja. Existuje viacero techník na testovanie softvéru. Medzi statické testovanie zaraďujeme recenzie, návody alebo kontroly. Naopak, pri dynamickom testovaní ide o naprogramovaný kód spoločne s testovacími údajmi. Statické testovanie sa v praxi často vynecháva. Dynamické testovanie zvyčajne prebieha až na finálnom produkte, ale podľa mňa by sa mala dynamicky testovať každá dokončená funkcionalita. Taktiež je možné testovať softvér z pohľadu postupu. Sú známe tri postupy testovania softvéru:

- Biela skrinka: tester má k dispozícii celý zdrojový kód programu, tým pádom vie presne zistiť, ako program pracuje.
- Čierna skrinka: tester vie ako program pracuje len na základe špecifikácie funkcionality.
- Šedá skrinka: zahŕňa v sebe znalosť zdrojového kódu na účely navrhovania testovacích prípadov na úrovni používateľa alebo čiernej skrinky.

Testovanie formou bielej skrinky býva miestami náročné, pretože je potrebné sa vedieť orientovať v zdrojovom kóde. Taktiež nie je jednoduché vytvoriť vstupy pre testovanie a určiť ich výstupy, či sú správne. Pri tejto forme testovania sa nedá odhaliť chýbajúca funkcionalita systému.

Z pohľadu testovania úrovne softvéru sa testujú funkcionálne ale taktiež aj nefunkcionálne vlastnosti softvéru. Najznámejšie sú:

- Jednotkové testovanie: pomocou napísaných jednotkových testov sa overuje funkčnosť určitých funkcií v systéme.
- Integračné testovanie: slúži na odhalenie chýb v rozhraní a interakcie medzi integrovanými komponentmi.
- Regresné testovanie: zameriava sa na opätovné testovanie systému, či zmeny nespôsobili nepriaznivé chovanie softvéru.
- Akceptačné testovanie: overuje sa chod systému so zákazníckymi požiadavkami.
- Alfa testovanie: testujú ho ľudia v rámci organizácie, ktorí tento softvér vyžadujú a podávajú o ňom správy.
- Beta testovanie: princípom rovnaké ako alfa testovanie, len testovanými osobami sú ľudia mimo organizácie.
- Výkonnostné a záťažové testovanie: softvér je kontrolovaný na výkonnostné požiadavky a korektné správanie pri maximálnom zaťažení.
- Používateľské testovanie: skúma sa, či je softvér ľahko naučiteľný a ovládateľný.

Ak prehlásime, že softvér je kvalitný na základe úspešného testovania, pričom bol testovaný viacerými testami (biela alebo čierna skrinka, integračné testovanie a pod.),

pričom v softvéri sa nevyzná nikto iný ako sám vývojár, tak to nie je pravda. Vždy je potrebné vykonať testy z každého druhu.

Vplyv vývoja riadeného testami na kvalitu softvéru

Až nedávno začali tímy zaoberajúce sa vývojom softvéru široko využívať vývoj riadený testami (test-driven development). Táto testovacia technika funguje tak, že pre každý malý kúsok programátorom vytvorenej funkcie sa musí najskôr vytvoriť jednotkový test (unit test). Programátori musia potom vytvárať kód, ktorý spĺňa jednotkový test. Toto núti programátorov premýšľať o mnohých aspektoch funkcie pred jej vytvorením. Poskytuje tiež bezpečné aktualizácie kódu bez toho aby programátor neporušil význam existujúcej funkcie. Programátori vytvárajú kód aby splnili tieto testy, a takto splnené testy sa ukazujú zainteresovaným stranám, že dodávaný kód spĺňa ich očakávania[4]. Dokáže naozaj vývoj riadený testami zlepšiť kvalitu softvéru?

Pracoval som doposiaľ v dvoch programátorských firmách, pričom jedna z nich používala vývoj riadený testami. Hneď môžem podotknúť, že firma, čo využívala túto metódu bola pomaly dvakrát tak väčšia ako firma, ktorá využívala klasické testovacie metódy ako biela či čierna skrinka. Na základe mojich skúseností tvrdím, že vývoj riadený testami produkuje kód s ďaleko menšími funkčnými chybami a exponenciálne stúpa pravdepodobnosť splnenia očakávania u zúčastnených strán v porovnaní s kódom vyprodukovanými konvenčnými programovacími technikami. Neustále si kladiem otázku, prečo sa viac firiem, keď už nie každá firma, neorientuje na vývoj riadený testami?

Je ťažké naučiť sa vývoj riadený testami. Problémom je napísať účinný jednotkový test pre nenaprogramovaný kód. Je odporúčané vývoj riadený testami testovať na dokonale čistom kóde, kde sa môže človek skutočne zamerať na funkčné testovanie. Väčšina chýb vzniká kvôli nedorozumeniu funkčných požiadaviek. Mnohokrát sa jednotkové testy vytvárajú spoločne so zákazníkom. Programátor môže použiť tieto testy aj na lepšie pochopenie a vytvorenie si obrazu pre správne návrhy funkcií. Všetky tieto aktivity napomáhajú produkovať čistý zdrojový kód vyhovujúci zákazníkovi. Ešte som sa nestretol s prípadom, keď niekto začne používať jednotkové testy aby sa vrátil k starším metódam.

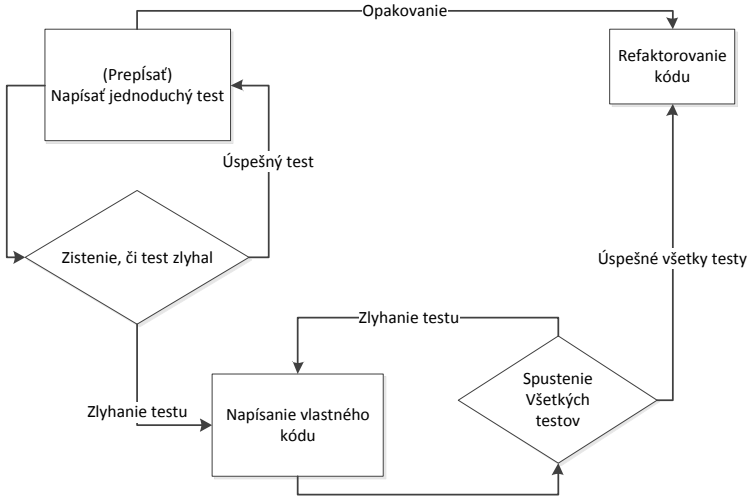
Vývojový cyklus vývoja riadeného testami

Postup vytvárania softvéru pomocou techniky vývoja riadeného testami vyzerá nasledovne [3]:

1. napísať jednoduchý test – prvým krokom pri tvorbe vývoja riadeného testami je vytvoriť test, ktorý overuje kód pre funkcionalitu, či spĺňa to, čo sa od neho očakáva. Tieto testy prakticky predstavujú definíciu požiadaviek na funkcionalitu. Vývojár sa napísaním takýchto testov presvedčí, že dostatočne presne rozumie požiadavke na funkcionalitu, a tým pádom eliminuje nedorozumenia pri implementácií. Mnohokrát býva problém aj pre skúseného programátora vytvoriť test na neexistujúci kód.

2. spustiť všetky testy a uistiť sa, že novo pridané zlyhajú – po dopísaní všetkých nových testov musia všetky končiť neúspechom, pretože ešte neexistuje kód, ktorý by ich úspešné dokončenie zaručil. Tento krok slúži ako akási sebakontrola, ktorá vylučuje možnosť, že by nová funkcionalita bola dokončená skôr akoby začala samotná implementácia. Ak niektorý z testov skončí úspechom, to znamená, že navrhovaná funkcionalita už v systéme existuje alebo vytvorený test je chybný.
3. napísanie vlastného kódu – v tomto kroku sa píše kód na všetky novovytvorené neúspešne prejdené testy. Jedná sa o rýchlo napísané kúsky kódu, v ktorých ide len o to, aby pri spustení novovytvorené testy prešli. Nekladie sa tu cieľ dosiahnuť čo najefektívnejší a najelegantnejší kód. Ide len o jednoduché splnenie testov, pretože efektívnosť a elegancia sa rieši v ďalších krokoch.
4. prechod kódu automatickými testami – ak kód napísaný v predchádzajúcom kroku prejde vyhodnotením testu ako úspešný, znamená, že sú v ňom plne definované požiadavky a môže sa prejsť na posledný krok.
5. refaktoring – rieši sa tu efektivita a elegancia napísaného kódu. Refaktoring kódu môže prebiehať aj automaticky za pomoci rôznych nástrojov. Odstraňujú sa duplicitné kódy a celkovo sa upravuje kód do čo najideálnejšej podoby. Opätovné spustenie automatických testov v tomto kroku umožňuje neustále overovanie, či zásahy do napísaného kódu nevniesli chyby do funkcionality.

Opakovanie - predchádzajúce kroky sa ďalej opakujú (Obr. 2). Na základe rôznych vstupov je opäť definovaná funkcionalita, ktorá sa vyjadrí pomocou automatických testov, ktoré znova pri prvom spustení neprejdú. Následne je znovu napísaný kód za účelom len splnenia testu, a ak je test úspešné splnený nasleduje refaktoring kódu. Tieto prírastky by mali byť čo najmenšie, približne 1-10 riadkov kódu medzi opätovným testovaním. Ak v ďalších krokoch skončí niektorý z predchádzajúcich testov neúspešne, znamená to návrat o pár krokov späť a zmena kódu.



Obr. 2. Grafická reprezentácia vývojového cyklu pre vývoj riadený testami.

Medzi najväčšiu výhodu vývoja riadeného testami radím neustále overovanie funkcionality systému na základe požiadaviek. Ak nejakým zásahom do kódu porušíme akúkoľvek funkcionalitu, tieto automatické testy ju okamžite odhalia. Podobná situácia vzniká pri zmene alebo doplnení funkcionality. Potom stačí pozmeniť alebo pridať nový test. Keďže tento prístup skladá program z malých častí, je jednoduché sa v ňom vyznať a spraviť potrebné zmeny.

Tak ako všade, tak aj vo vývoji riadenom testami sú isté nevýhody. Tento prístup je ťažko použiteľný v situáciách, kedy funkčné testy sú plne požadované na určenie úspechu alebo neúspechu. Ako príklad môže uviesť užívateľské rozhranie alebo programy spolupracujúce s databázovým systémom. Testy, ktoré obsahujú tvrdo nakódovanú chybu alebo reťazec sú náchylné na udržiavanie.

Záver

Každý jeden z vybraných troch modelov procesu vývoja softvéru zabezpečuje dostatočnú kvalitu. Na prvý pohľad sa môže zdať, že ide o totožné metódy. Avšak RUP a MSF patria medzi kvalitnejšie procesy vývoja softvéru, pretože obsahujú viac techník na zabezpečenie kvality ako pri XP. XP je zamerané na menšie projekty, kde sa programuje vo dvojici. Testovanie dohliada na detegovanie chýb v softvéri, splnenie požiadaviek zo strany zákazníka a hodnotí kvalitu softvéru. Po analýze testovacích techník som sa rozhodol zamerať na vývoj riadený testovaním, pretože je najvhodnejší pre náš tímový projekt. V kombinácií s XP by mohol projekt dosiahnuť najlepšiu kvalitu.

Použitá literatúra

1. Akingbehin, K.: Towards Destructive Software Testing. In: *Computer and Information Science,* Honolulu (2006), 374-377.
2. Amis, M., et al.: *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Standards Board (1992).
3. Beck, K.: *Test Driven Development: By Example*. Addison-Wesley, Boston, 2002.
4. Crispin, L.: Driving Software Quality: How Test-Driven Development Impacts Software Quality. In: *IEEE Software*, Vol. 23, No. 6 (2006), 70-71.
5. Hetzel, W.C., Hetzel, B.: *The Complete Guide to Software Testing, 2nd edition*. John Wiley & Sons, Inc. New York, 1991.
6. Tassey, G.: The Economic Impacts of Inadequate Infrastructure for Software Testing. In: *Research Triangle Institute for the National Institute of Standards and Technology* (2002), 95-101.

Annotation

Despite the quality, you should always test

Diversity and complexity of software is increasing day by day, therefore, must be provide software quality. Software quality is considered more attributes of quality, whether it's performance and

usability for end-user or high stability and reuse. Emphasis on improving the quality of the software development system and avoids confusion in the actual implementation. Testing is compared to software quality assurance focused on "detection". When testing software quality reveals the shortcomings and errors developed by the software, therefore it is necessary to test the software, although it seems that it is not necessary. Currently, there are many different methods of how to properly test the software quality. The essay focuses on quality assurance and software convinces you that the development of controlled tests, saving valuable time and money by comparing with software testing after the completion of implementation.