# VPLÝVA TESTAMI RIADENÝ VÝVOJ NA KVALITU SOFTVÉRU?

*Testovať, testovať, testovať.*

*Marek Račev*

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
marek.racev[zavináč]gmail[.]com

***Abstrakt.*** *S rozvojom informačných technológií a neustále rastúcim množstvom požiadaviek je čoraz náročnejšie vyvíjať kvalitný softvér. Mnohé programátorské tímy si na zabezpečenie kvality osvojujú agilné metódy vývoja softvéru. V tejto eseji sa zaoberám jednou z kľúčových praktík extrémneho programovania – testami riadeným vývojom. Základným princípom tohto prístupu je písanie automatizovaných testov pred vývojom samotného úžitkového kódu v malých rýchlych iteráciách. Hlavná otázka, ktorú sa snažím v eseji zodpovedať je, či testami riadený vývoj vplýva na kvalitu softvéru v zmysle kvality doručeného softvéru a internej kvality kódu. Špeciálnu pozornosť venujem výhodám a nevýhodám tohto prístupu a vhodnosti jeho použitia v závislosti od typu vyvíjaného projektu.*

***Kľúčové slová:*** *kvalita, testami riadený vývoj, testovanie, extrémne programovanie*

## Úvod

S prudkým rozvojom informačných technológií sú na vyvíjaný softvér neustále kladené nové požiadavky. Pre softvérové spoločnosti je čoraz náročnejšie vytvárať kvalitné systémy, najmä ak chcú udržať krok s konkurenčnými firmami a byť schopné rýchlo a flexibilne reagovať na meniace sa prostredie a nové trendy v oblasti informatiky.

Čo však máme na mysli, ak hovoríme o kvalite softvéru? Pod pojmom kvalita si zrejme každý predstaví niečo iné, no kvalita softvéru je vo všeobecnosti daná mierou

splnenia funkcionálnych a nefunkcionálnych požiadaviek [2]. Funkcionálne požiadavky predstavujú množinu čŕt (angl. features), ktoré vyvíjaný softvér má, resp. nemá obsahovať, zatiaľ čo nefunkcionálne požiadavky vymedzujú jeho vlastnosti. Základný rámec umožňujúci softvérovým inžinierom definovať kvalitu softvéru – štandard ISO 9126, definuje 6 hlavných charakteristík kvalitného softvéru: funkcionalita, spoľahlivosť, použiteľnosť, efektívnosť, udržovateľnosť a portabilita.

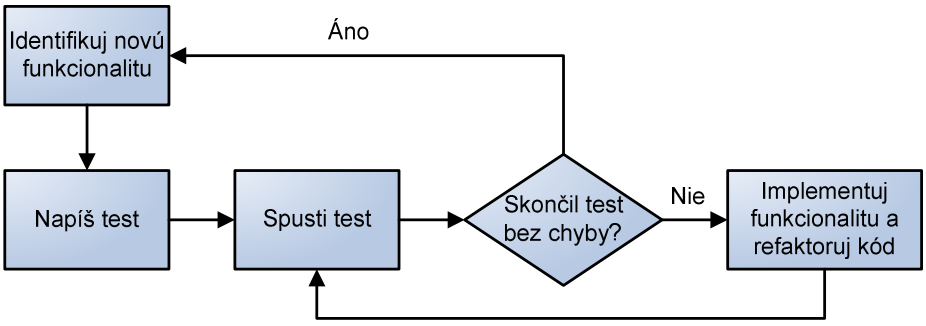
Zabezpečovanie kvality je pri rozmeroch a komplexnosti súčasných softvérových systémov veľmi aktuálnym problémom. Jedným zo spôsobov, ako môžeme významne prispieť k zlepšeniu kvality je zvoliť si vhodný prístup k vývoju a testovaniu softvéru pre aktuálny projekt. Mnohé programátorské tímy s cieľom zlepšenia kvality ich softvérových produktov postupne prechádzajú od tradičných prístupov k moderným technikám (napríklad k agilným metódam vývoja softvéru).

V tejto eseji predstavím agilnú metódu nazývanú testami riadený vývoj a pokúsim sa čitateľovi priblížiť jej základné výhody a nevýhody a zhodnotiť vhodnosť jej použitia v závislosti od vyvíjaného projektu. V ďalšej časti sa budem venovať vplyvu tejto metódy na kvalitu softvéru, predovšetkým z pohľadu internej kvality kódu a kvality dodaného softvéru, pričom sa budem opierať o výsledky experimentov softvérových výskumníkov.

## Testami riadený vývoj

Testami riadený vývoj (angl. Test-Driven Development; ďalej TDD) je jednou z kľúčových stratégií extrémneho programovania, ktorá si neustále získava čoraz väčšiu pozornosť. Hlavným princípom tohto prístupu je písanie automatizovaných testov pred vývojom samotného úžitkového kódu v malých rýchlych iteráciách.

Jedna iterácia predstavuje krátky vývojový cyklus, v ktorom programátor najskôr napíše automatizovaný test, ktorý definuje novú funkcionalitu. Následne vytvorí kód, ktorý v tomto teste uspeje a nakoniec ho podľa potreby „vyčistí“ (angl. refactor). Pod pojmom refaktoring chápeme proces modifikácie programu za účelom zlepšenia jeho štruktúry a redukcie jeho zložitosti [5]. Takáto reštrukturalizácia a reorganizácia kódu by však nemala viesť k zmene jeho pôvodnej funkcionality.




**Obr. 1.** Základný proces testami riadeného vývoja [5].

Základný proces TDD môžeme vidieť na obr. 1. Kroky tohto procesu sú nasledovné [1]:

1. Proces začína identifikovaním novej črty s požadovanou funkcionalitou. Za bežných okolností sa jedná o malý kúsok kódu, ktorý obsahuje iba niekoľko riadkov.
2. Programátor napíše pre túto črtu automatizovaný test (t.j. test, ktorý je vykonateľný a ohlási, či nastala chyba alebo nie).
3. Následne programátor spustí test spolu so všetkými doteraz implementovanými testami. Nový test musí nevyhnutne zlyhať, pretože bol vytvorený ešte pred samotnou implementáciou črty. V prípade, že test nezlyhá, je buď chybný alebo črta pre ktorú bol test naprogramovaný už existuje.
4. Po overení funkčnosti nového testu programátor napíše minimálny kód, ktorý prejde týmto testom. Kód býva v tejto fáze častokrát nedokonalý. Dôležité je, aby bol naprogramovaný iba s cieľom prejsť novým testom, teda nemal by predvídať žiadnu ďalšiu funkcionalitu.
5. Ak zbehli všetky testy bez chyby, programátor si môže byť istý, že implementovaný kód spĺňa všetky testované požiadavky.
6. Pokiaľ je to potrebné, programátor môže refaktorovať svoj kód a odstrániť prípadné duplicity. Znovuspustením testov sa programátor môže presvedčiť, že úprava kódu nezmenila jeho pôvodnú funkcionalitu.
7. V prípade, že upravený kód prejde všetkými testami, pokračujeme krokom 1.

## Je vhodné použiť testami riadený vývoj?

Predstavte si, že by sa vás niekto snažil presvedčiť, že základné koncepty vývoja softvéru, ktoré celý život používate sú nevhodné a v podstate nesprávne. Čo by vás prinútilo zmeniť váš názor? Paradigma TDD tvrdí, že základný cyklus implementácie a následného testovania, pomocou ktorého sa ubezpečujeme, že kód robí to čo má (niečo, čo nám bolo „vtĺkané“ do hlavy odkedy sme sa začali učiť programovať) nie je najefektívnejší prístup k tvorbe kvalitného softvéru. TDD nahrádza tradičný cyklus „najprv kód, potom test“ zavedením opačného postupu.

Písanie testov pred samotným úžitkovým kódom núti programátora zamýšľať sa nielen nad rozhraním softvéru (mená funkcií a metód, návratové typy, parametre, výnimky) ale aj nad jeho správaním (napríklad aké sú očakávané výsledky pre daný vstup) [1], čo vytvára vhodný základ pre vznikajúci návrh systému. TDD ale nie je všeliek a jeho používanie nám kvalitný návrh nezabezpečí. TDD však má vo všeobecnosti potenciál návrh zlepšovať, pretože nabáda programátorov k písaniu voľne viazaných súčiastok a k zamysleniu sa nad návrhom úplne odlišným spôsobom ako pri tradičnom vývojovom cykle. Mnoho ľudí si vôbec neuvedomuje, že TDD nie je iba prístup k testovaniu, ale aj prístup k vývoju a návrhu softvéru.

Ak by sa ma niekto spýtal, prečo je vhodné používať práve prístup TDD, nemusel by som nad odpoveďou dlho váhať. Podľa môjho názoru, ale aj podľa viacerých autorov [1, 3, 4], programátori, ktorí si osvojili techniky TDD, produkujú čistejší a flexibilnejší kód s menším množstvom chýb (ako uvidíme ďalej, s týmto tvrdením sú v zhode aj výsledky väčšiny prípadových štúdií zaoberajúcich sa dopadmi TDD na kvalitu kódu). Vďaka

rýchlej spätnej väzbe automatizovaných testov sú programátori sebavedomejší a majú väčšiu dôveru v korektnosť svojho kódu.

Prečo mnohí programátori „nečistia“ svoj kód? Obávajú sa, že po reorganizácii a reštrukturalizácii kód prestane fungovať. Vo svojej praxi som sa často stretol s typickým postojom programátorov: „Nebudem predsa opravovať to, čo funguje.“ Pri použití metódy TDD sa však úpravy kódu nemusíme obávať, pretože vďaka automatizovaným testom môžeme takmer okamžite skontrolovať, či vykonané zmeny neovplyvnili funkcionalitu softvéru definovanú týmito testami. Dokonca aj komplexné zásahy a úpravy zdrojového kódu so sebou prinášajú iba veľmi malé riziko, ak sa realizujú postupnosťou malých zmien medzi, ktorými necháme zbehnúť testy [3].

Ako sa zoznamujete s novými knižnicami, ktoré používate pri vývoji softvéru? Prečítať si dokumentáciu je iste užitočné, no skutočné pochopenie častokrát prichádza až po preštudovaní a preskúšaní príkladných zdrojových kódov. Každý test softvérovej súčiastky, ktorý pri používaní metódy TDD napíšeme, predstavuje izolovaný kúsok kódu, ktorý vysvetľuje ako niektorá časť celého systému funguje [3]. Ak napríklad chceme vedieť ako zavolať určitú funkciu, máme k dispozícii test, ktorý ju volá. Jednoducho povedané, testom môžeme opísať čokoľvek, čo chceme robiť so systémom. Dokumentovanie pomocou testov softvérových súčiastok so sebou prináša niekoľko zásadných výhod: sú formálne a sú napísané v jazyku, ktorý je pre programátorov zrozumiteľný; testy sú neustále vykonávané, a preto sú vždy (na rozdiel od písanej dokumentácie) v súlade s aktuálnou verziou systému.

Za jeden z najpresvedčivejších argumentov, prečo je vhodné použiť TDD, osobne považujem schopnosť včasnej detekcie a veľmi presnej lokalizácie miesta výskytu chyby. Ak striktne postupujeme podľa procesu TDD (uvedeného v predchádzajúcej kapitole), vždy nás od najbližšieho spustenia testov delia iba minúty. Vďaka prepracovanému systému TDD vieme novovzniknuté chyby lokalizovať veľmi efektívne – chyba sa predsa musí nachádzať v poslednej implementovanej súčiastke. V skutočnosti je situácia o niečo komplikovanejšia, pretože s pribúdajúcim množstvom testov narastá aj čas potrebný na ich vykonanie, a preto nemôžu byť do procesu testovania zakaždým zahrnuté úplne všetky testy. Kompletné spustenie testov sa preto vykonáva v dlhších časových intervaloch v závislosti od veľkosti projektu.

TDD samozrejme nie je dokonalá metóda a prináša so sebou aj niekoľko zásadných nevýhod. Testy aj úžitkový kód tej istej softvérovej súčiastky zvyčajne vytvára ten istý človek, a preto môžu oba tieto komponenty obsahovať tú istú chybu. Napríklad ak chceme v programe uskutočniť výpočet s operáciou delenia, mali by sme skontrolovať, či nedelíme čísla nulou. Ak na to zabudneme napísať príslušný test, kód ktorý by kontroloval hodnotu menovateľa nikdy nebude zahrnutý v programe. Programátor si však musí písať vlastné testy, pretože nemôže čakať 20-krát denne na niekoho iného, kým daný test napíše [1].

Naučiť sa používať TDD je pomerne komplikované. Ide predovšetkým o to, naučiť sa napísať test pre softvérovú súčiastku, ktorá (v čase písania testu) neexistuje. Zvládnutie tejto metódy vyžaduje veľa času a disciplíny, a je vhodná skôr pre skúsenejších programátorov. TDD nie je vhodný prístup pre použitie v ľubovoľnom projekte a na ľubovoľný účel. TDD sa napríklad nehodí na komplexné funkčné testovanie (testovanie používateľského rozhrania, programov pracujúcich s databázami a podobne). Túto

metódu je však ideálne použiť pre menšie vývojové tímy (pozostávajúce z 2 až 10 programátorov), ktoré pracujú na malých a stredne veľkých projektoch.

## Aké sú dopady testami riadeného vývoja na kvalitu softvéru?

Na kvalitu softvéru sa môžeme pozerať z viacerých pohľadov a na rôznych úrovniach abstrakcie. V tejto časti sa budem podrobnejšie venovať otázke, ako vplýva TDD na kvalitu doručeného softvéru a na internú kvalitu kódu, pričom sa budem opierať o výsledky prípadových štúdií uskutočnených výskumníkmi v priebehu posledných rokov. Cieľom týchto štúdií bolo empiricky overiť hypotézy spojené s vplyvom TDD na kvalitu softvéru v porovnaní s tradičnými prístupmi pri rovnakých alebo podobných podmienkach.

### Kvalita doručeného softvéru

Tab. 1 sumarizuje na základe výsledkov 21 štúdií dopady TDD na kvalitu doručeného softvéru v porovnaní s tradičnými metódami. Pri vyhodnocovaní úspešnosti porovnávaných prístupov boli použité rôzne metriky, ako napríklad percentuálny podiel úspešnosti testov finálneho produktu, hustota chýb, počet odhalených chýb v rámci jedného testu, celkové úsilie potrebné na zabezpečenie kvality finálneho produktu.

Výsledky pilotných a pracovných štúdií ukazujú jasnú dominanciu úspešnosti TDD. Riadené experimenty vykonávané prevažne v akademickom prostredí však podávajú nejednoznačné výsledky (odchýlka bola pravdepodobne spôsobená malou skúsenosťou študentov s TDD). Myslím si, že TDD naozaj zlepšuje kvalitu hlavne vďaka skorej detekcii chýb a jednoduchého spôsobu ich lokalizovania. TDD taktiež vedie programátorov, aby sa zamýšľali nad návrhom, a tým predchádzali vzniku chýb spôsobených zlým návrhom.

**Tab. 1.** Prehľad výsledkov štúdií skúmajúcich vplyv TDD na kvalitu dodaného softvéru [4].

| Typ štúdie | TDD vplýva na kvalitu pozitívne | TDD neovplyvňuje kvalitu | TDD vplýva na kvalitu negatívne |
|---|---|---|---|
| Riadený experiment | ✓ | ✓✓✓ | ✓✓ |
| Pilotná štúdia | ✓✓✓✓✓✓ | ✓✓ | |
| Pracovná štúdia | ✓✓✓✓✓✓ | ✓ | |
| Celkový počet | 13 | 6 | 2 |

### Interná kvalita kódu

Tab. 2 zobrazuje spracovanie 14 štúdií, ktorých výsledky sa týkajú kvality kódu. Znak „★“ reprezentuje štúdiu, ktorá vyhodnotila metódu TDD za úspešnejšiu iba pri niektorých z použitých metrík. Pri vyhodnocovaní má takáto štúdia polovičnú váhu. Medzi skúmané metriky patria viazanosť a súdržnosť softvérových súčiastok, zložitosť kódu a organizovanosť kódu, ktorá je zameraná na veľkosť modulov a počet riadkov kódu potrebných na implementáciu súčiastky.

Nejednoznačnosť výsledkov všetkých troch typov štúdií ma veľmi prekvapila. Podľa procesu TDD sa po implementácii každej softvérovej súčiastky aplikuje refaktoring, ktorý vedie k zrozumiteľnejšiemu, organizovanejšiemu a urdžovateľnejšiemu kódu. Podľa Shull

et al. [4] však TDD vplýva priaznivo na kvalitu kódu a výsledky štúdií preto vnímajú skepticky. Tímy, ktoré si osvojili TDD produkujú čistejší kód s voľnejšie viazanými súčiastkami, ktorý sa jednoduchšie udržiava a rozširuje.

**Tab. 2.** Prehľad výsledkov štúdií skúmajúcich vplyv TDD na internú kvalitu kódu [4].

| Typ štúdie | TDD vplýva na kvalitu pozitívne | TDD neovplyvňuje kvalitu | TDD vplýva na kvalitu negatívne |
|---|---|---|---|
| Riadený experiment | ✓ | ✓✓ | |
| Pilotná štúdia | ✓★★★ | ✓✓ | ✓★★★ |
| Pracovná štúdia | ✓✓✓ | | ✓ |
| Celkový počet | 6,5 | 4 | 3,5 |

## Záver

Hoci nemám s testami riadeným vývojom rozsiahle praktické skúsenosti, myslím si, že sa jedná o veľmi perspektívny prístup k vývoju softvéru. Som presvedčený, že testami riadený vývoj má obrovský potenciál zlepšovať nielen kvalitu doručeného softvéru, ale aj zdrojového kódu.

Písanie testov pred samotným úžitkovým kódom so sebou prináša veľké množstvo výhod, medzi ktoré patrí zlepšenie softvérového návrhu, rýchlosti detekcie, lokalizácie a opravy chýb, tvorba čistého, dobre organizovaného a udržovateľného kódu a taktiež zaujímavá forma autodokumentácie.

Testami riadený vývoj nepredstavuje vhodný prístup pre každý projekt a nemali by sme sa ho snažiť aplikovať za každých okolností. Napriek tomu si myslím, že pri vhodnej téme tímového projektu môže byť použitie tejto metódy zaujímavou skúsenosťou, ktorá by mohla tímu pomôcť produkovať spoľahlivý softvér s prehľadným a rozšíriteľným kódom.

## Použitá literatúra

1. Beck, K.: *Test-Driven Development: By Example*. Addison-Wesley, 2002.
2. Bieliková, M.: *Softvérové inžinierstvo. Princípy a manažment*. 1. vyd. Bratislava: STU, 2000. 220 s. ISBN 80-227-1322-8.
3. Martin, R. C.: *Professionalism and Test-Driven Development*. IEEE Software, s. 32-36, Máj/Jún, 2007.
4. Shull, F., Melnik, G., Turhan, B. et al.: *What Do We Know about Test-Driven Development?*. IEEE Software, s. 16–19, November/December, 2010.
5. Sommerville, I.: *Software Engineering*. 9. vyd. Addison-Wesley, 2011.

## Annotation

*Does test-driven development influence software quality?*

*With advancement in information technologies and increasing amount of requirements it is more and more difficult to develop high-quality software. Many development teams are adopting agile methods of software development in order to ensure quality. In this essay I discuss one of the core extreme programming practices – the test-driven development. The fundamental principle of this approach is automated tests writing prior to developing of functional code in small, rapid iterations. The main question that I am trying to answer is whether test-driven development has an impact on software quality in terms of delivered software quality and internal code quality. A special focus is given to pros and cons of this approach and the suitability of its use depending on type of project.*