JE MOŽNÉ VYTVÁRAŤ KVALITNÝ SOFTVÉR BEZ TESTOVANIA?

Testovanie nemôže preukázať, že v programe nie sú chyby. Môže iba ukázať, že tam chyby sú.

Peter Petriľák

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

Abstrakt. *S pojmom kvalita sa stretávame dennodenne vo všetkých sférach života. Aj v oblasti softvérového inžinierstva má tento pojem strategický význam. Ale dosiahnutie kvality v softvéri je náročný proces. V súčasnosti existuje veľa spôsobov nato ako to docieliť. Jedným z nich je aj vylepšovanie zdrojového kódu. V eseji sa zameriavame na spôsoby, akými vývojári môžu vylepšiť zdrojový kód a tak zvýšiť kvalitu softvéru. Niekedy sa stretávame s postojmi, že dodržiavanie zásad, ktoré zdokonaľujú zdrojový kód, je zárukou pre kvalitný softvér. Takéto uvažovanie môže byť zavádzajúce. Na životnom cykle tvorby softvéru sa podieľa viacero fáz a jednou z nich je aj testovanie. V eseji predostrieme náš názor, či je možné dosiahnuť kvalitný softvér bez tejto fázy.*

Kľúčové slová: *softvér, kvalita softvéru, komentáre, dokumentácia, refaktoring, zdrojový kód, testovanie*

Úvod

V tejto eseji sa zaoberáme otázkou čo sa v skutočnosti skrýva za pojmom softvérová kvalita. Odovzdať klientovi softvérový produkt, ktorý nespĺňa jeho požiadavky kvality by bola blamáž. Aby k tomu nedochádzalo, musia softvérové firmy venovať čo najviac úsilia splneniu všetkých požiadaviek klienta. Výsledkom je spokojnosť nielen zákazníka ale aj používateľov, pre ktorých je daný softvér určený.

Z článku [3] vyplýva, že slabé testovanie softvérových systémov znižuje spoľahlivosť softvéru, ktorá následne negatívne ovplyvňuje jeho kvalitu. S cieľom dosiahnutia vyššej efektívnosti testovania, sa musia softvérové spoločnosti zamerať na lepšiu kultúru vývoja softvéru. Rôzne testovacie techniky, metódy a nástroje môžu testovaniu pomôcť, avšak je najmä na manažmente a na ľuďoch, ktorí sa podieľajú na vývoji softvéru teda najmä vývojároch, aby bol softvér, ktorý je určený na testovanie, čo možno najkvalitnejší. V tejto eseji sa zameriavame na to, ako môžu vývojári dosiahnuť čo možno najkvalitnejší softvér a aké sú praktiky, napomáhajúce zlepšeniu softvéru, ktoré by mali používať pri programovaní.

Zvýšenie kvality z programátorského hľadiska

Keďže pojem softvérová kvalita je veľmi široký, nedá sa v takejto eseji sústrediť na jeho problém celoplošne. Preto sa v tejto eseji budeme snažiť popísať ako sa môžu pričiniť ku kvalite softvéru tí, ktorí stoja za jeho zdrojovým kódom, čiže programátori. Rozhodli sme sa preto aj z toho dôvodu, že už máme tiež isté skúsenosti z tejto pozície.

Je veľa spôsobov alebo štandardov, ktorými by sa mali programátori riadiť a ktoré odporúčajú starší, skúsenejší kolegovia mladším. Je bežné, že viacero ľudí pracuje na jednom projekte a pracujú nad jedným zdrojovým kódom. Stávajú sa situácie, kedy jeden programátor pracuje s kódom, ktorý naprogramoval iný a vtedy je veľmi dôležité aby sa navzájom vedeli dorozumieť a čo najlepšie a najrýchlejšie pochopiť kód toho druhého. Preto sa v tejto eseji zaoberáme problematikou dokumentácie kódu resp. písaním zrozumiteľného kódu a komentárov. Pri častých zmenách dochádza k ovplyvňovaniu štruktúry a funkcionality softvéru. Tento vplyv zmien sa väčšinou prejaví ihneď, ale niekedy sa môže prejaviť v neskorších fázach vývoja softvéru a môže byť aj negatívny. Preto je taktiež potrebné sa zamyslieť nad problematikou dlhodobého vplyvu zmien pri vývoji softvéru. Obidve tieto oblasti by sa dali istým spôsobom zaradiť aj k problematike refaktoringu, ktorý je takisto popísaný v tejto eseji.

Komentáre

Podľa Jefa Raskina[2] je dôkladné používanie internej dokumentácie jedným z najviac prehliadaných spôsobov vylepšovania kvality softvéru a zrýchľovania implementácie. Presadzovanie excelentnosti pri dokumentovaní kódu je zatiaľ nevyriešený problém pri vývoji softvéru. Niektoré z riešení sa zdajú byť dostatočne efektívne, avšak nie sú úplne udomácnené v kultúre programovania. Dokonca existujú vývojári, ktorí úplne odmietajú dokumentovanie programov, ako napríklad prívrženci extrémneho programovania, ktorí to považujú za zbytočné.

Inline komentáre nie sú veľmi vhodné a ja osobne ich ani nepoužívam. Ich nevýhodou je najmä to, že sú obmedzené iba na jeden riadok. Ba dokonca, keďže sa uvádzajú za kód v riadku, ktorý je často krát už veľmi odsadený od okraja, nie je veľký priestor pre text komentára a tak musia byť veľmi strohé a nemusia dostatočne jasne okomentovať daný problém. Niekedy môžu byť až na príťaž a na ich interpretáciu je najprv potrebné pochopiť kód. Preto sa pri reštrukturalizácii a aktualizácii kódu stáva, že sú nakoniec vymazané. Podľa mňa je najlepšie použiť celý riadok alebo aj viacero riadkov

pre napísanie podrobnejšieho komentára, ktorý výstižne opíše danú problematiku. Najmä pri metódach resp. funkciách, ktoré sú dlhé a reprezentujú v sebe zložitú logiku je lepšie si prečítať viacriadkový komentár a takto to rýchlo pochopiť, ako sa snažiť rozlúsknuť kód bez pomocných komentárov.

Je možné automatické generovanie dokumentácie zo zdrojového kódu. Avšak napriek tomu, že to môže na prvý pohľad znieť ako výhodné a rýchle riešenie ako vytvoriť dokumentáciu, nedosahuje postačujúcu kvalitu. Fundamentálny dôvod prečo nemôže byť automatické generovanie dokumentácie postačujúce, je to, že komentáre nedokážu vysvetliť prečo je program napísaný a nedokážu zdôvodniť použitie tej ktorej metódy. Nemôžu diskutovať použitie alternatívnych metodík alebo prístupov. Ako príklad uvediem komentár[2]: *„Ukázalo sa, že binárne vyhľadávanie je pomalšie ako Boyer-Moorov algoritmus, preto sme použili komplexnejšiu, ale rýchlejšiu metódu aj keď tento problém na prvý pohľad nevyzerá prístupný pre techniku vyhľadávania reťazca.“* Tento komentár nielenže menuje použitú techniku, ale dokonca aj opisuje, prečo nebol použitý jednoduchší prístup.

Podľa Donalda Knutha[4] je základný koncept písania najprv dokumentácie, potom vytvárania metód a popísania ich logiky kľúčom k vysoko kvalitnému komerčnému programovaniu. Prednostná, jasná a rozsiahla dokumentácia je kľúčovým aspektom pri vytváraní softvéru, ktorý môže pretrvať a prispôsobiť sa. Dokumentovanie zníži čas potrebný na vývoj softvéru a vyústi v lepšiu prácu a kvalitnejší výsledok.

Samo dokumentovateľný kód

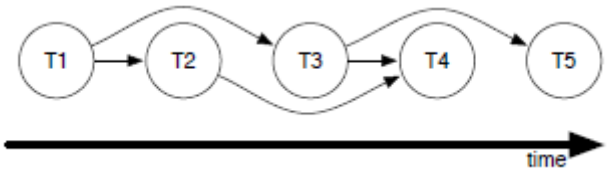
Keď sa povie slovné spojenie samo dokumentovateľný kód („*self-documenting code*“), máme na mysli používanie rôznych techník pri programovaní, ako napríklad jasné a zrozumiteľné pomenovanie názvov premenných. Namiesto použitia názvu premennej *n* alebo *pocet*, by bolo vhodnejšie použiť názov napríklad *pocetNazbieranychHrusiek*. Je to na zvážení každého programátora, či bude používať takýto spôsob označovania alebo nejaký vlastný, ale každopádne to zvyšuje zrozumiteľnosť názvov, či už premenných, modulov objektov alebo programov. Keď vidíme takýto názov, hneď nám je jasné čo to znamená a nie je potrebný ani dodatočný komentár.

Dlhodobý vplyv zmien

Softvérový vývoj je inkrementálny proces, pri ktorom sa používajú predošlé verzie softvérového produktu pre vytvorenie novšej verzie. Preto robia vývojári často zmeny v kóde s cieľom pridania novej funkcionality, opravy nájdených chýb alebo revízie zdrojového kódu. Hneď po zmene kódu sa môže štruktúra softvéru a jeho správanie zmeniť[1]. Veľa z týchto zmien má okamžitý vplyv na kvalitu a stabilitu softvéru. Avšak niektoré z nich sa môžu prejaviť až v oveľa neskoršej dobe. Schopnosť identifikovať zmeny s dlhodobým vplyvom na softvér nám umožňuje lepšie pochopiť postupný vývoj zmien v softvéri a tak lepšie odhadnúť budúce rozhodnutia a zmeny v kóde.

Ovplyvnenie procesu vývoja počas určitej periódy času znamená vplývať na neskoršie rozhodnutia. Na odhalenie dlhodobého vplyvu je potrebné zaznamenávanie alebo rekonštrukcia závislostí medzi jednotlivými zmenami. Čím vyššie je číslo zmeny, ktorá závisí na *Ti*, tým dlhodobejší je vplyv *Ti*. Na grafe závislosti transakcií znázornenom

na obrázku č. 1 je zobrazený príklad zmien. Vrcholy *T1 – T5* znázorňujú jednotlivé zmeny vykonané v čase a hrany medzi nimi reprezentujú závislosť týchto zmien. Z grafu je vidieť, že vplyv zmeny T2 je malý, keďže iba jedna z neskôr aplikovaných zmien na nej závisí. Skúmanie informácií o zmene T4 nám umožňuje vyvodiť záver o vplyve T2 na kvalitu a výkon procesu vývoja v neskorších fázach.



Obr. 1: Graf závislosti transakcií

Rozdiel medzi dvomi revíziami kódu je jednoduché zistiť pomocou softvéru na zisťovanie rozdielov. Avšak identifikovanie skutočného dopadu na správanie softvéru si vyžaduje oveľa podrobnejšiu analýzu.

Neustále zmeny v požiadavkách vyvolávajú časté zmeny kódu. Je samozrejmé, že zmeny kódu navzájom na sebe závisia. Pochopenie zložitosti vývoja softvéru a schopnosť sledovať na základe akého rozhodnutia sú moje zmeny kódu založené, poskytujú základné informácie, ktoré vymedzujú kvalitu zmeny. V práci[1] autor využíval na preskúmanie dlhodobého vplyvu zmien na programy grafy závislosti transakcií. Jeho záverom je, že tieto grafy môžu byť použité na odhad a meranie dlhodobého vplyvu zmien. Meranie dlhodobého vplyvu zmien by podľa neho mohlo pomôcť zlepšiť odhadovanie účinku vytáraných zmien.

Refaktoring

Softvérové systémy musia podstupovať úpravy a vylepšenia, aby sa vyrovnali s narastajúcimi požiadavkami. Táto údržba spôsobuje úpadok ich kvality. Na vyhodnotenie spôsobu akým je ovplyvnená kvalita systému sa používajú rôzne metriky. Refaktoring je jedným z najdôležitejších a najpoužívanejších techník transformovania častí softvéru za účelom vylepšenia jeho kvality. Je definovaný ako proces zmeny softvérového systému spôsobom, ktorý neovplyvní jeho funkcionalitu, ale vylepší jeho vnútornú štruktúru. Je to disciplinovaný spôsob ako upratať kód, ktorý minimalizuje pravdepodobnosť zavedenia chýb. V podstate keď robíte reštrukturalizáciu, zlepšujete dizajn kódu potom čo bol napísaný. [6] Pri refaktoringu sa môže vziať zlý dizajn, dokonca chaotický kód a pretvoriť ho do dobre dizajnovaného a štruktúrovaného kódu. Množstvo dostupných nástrojov pre zbieranie softvérových metrík a automatizované používanie refaktoringu zjednodušuje údržbu softvéru. To znamená, že čoraz viac vývojárov a softvérových inžinierov používa refaktoring vedome na zvýšenie kvality softvéru.

Autori v práci[5] analyzovali zdrojové kódy v systémoch kontroly verzií populárnych open source systémoch a hľadali zmeny označené ako reštrukturalizácia. Skúmali ako sú softvérové metriky ovplyvnené týmto procesom, aby zistili či je refaktoring skutočne používaný open source komunitou na efektívne zlepšenie kvality.

Očakávané a skutočné výsledky sa často líšia a to dokázala aj táto štúdia. Aj keď vývojári používajú refaktoring s cieľom zvýšenia kvality, metriky naznačujú, že tento proces má niekedy opačný efekt. Výsledky ukazujú významnú zmenu niektorých ukazovateľov k horšiemu. V niektorých systémoch došlo k zníženiu súdržnosti niektorých tried, pri iných k zvýšeniu zložitosti. Aj keď sa nedá kvalita softvéru merať iba metrikami, tieto výsledky ukazujú, že buď proces reštrukturalizácie nedokáže zakaždým vylepšiť kvalitu softvéru viditeľným spôsobom alebo vývojári ešte nedospeli do stavu, kedy by refaktoring vedeli používať ako efektívny spôsob pre zvýšenie kvality vyvíjaného softvéru.

Táto štúdia bola však testovaná na malom množstve programov a preto sa podľa mňa nedá chápať ako všeobecne platná pre túto oblasť. Nakoniec aj autori v závere pripúšťajú, že je potrebné vykonať podobné overenie na omnoho väčšom množstve softvéru. Vtedy by bolo lepšie preskúmané aj to, ktoré spôsoby refaktoringu sú najúčinnejšie a ktoré metriky sú najvhodnejšie pre meranie kvality.

Refaktoringom k lepšiemu pochopeniu kódu

Aby bol kód ľahšie zrozumiteľný je potrebné urobiť niekedy zmeny. Ale prečo by sme mali chcieť aby bolo jednoduchšie ho pochopiť? Pretože ak sa dostane niekto iný k nášmu kódu a nie je to jasne naprogramované, môže mať dotyčný ťažkosti s jeho porozumením. Ak sa napríklad ocitneme v situácii, že nebudeme vedieť vyriešiť nejaký problém a príde nám na pomoc skúsenejší programátor, je dobré aby bol náš kód jasný a tak preňho ľahko pochopiteľný. Alebo môže nastať situácia, že sa po dlhom čase budeme musieť vrátiť ku kódu, ktorý sme dávno nevideli a aj keď pred časom sme mu rozumeli, teraz môžeme mať problém s jeho pochopením ak nie je dobre a výstižne napísaný. Preto môže refaktoring napomôcť k lepšej čitateľnosti kódu. Malé množstvo času vynaloženého teraz na refaktoring sa nám môže vrátiť a ušetriť oveľa viac času v neskorších fázach vývoja.

Tento spôsob môže fungovať aj opačným smerom. Ak niečomu programátor nerozumie môže sa to pokúsiť upraviť tak, aby to bolo jasnejšie vyjadrené nielen pre toho, kto to prvotne napísal ale aj pre ostatných. Avšak tu treba dbať obzvlášť veľký pozor nato, aby nedošlo k zmene funkcionality. Preto to treba vždy dobre otestovať a uistiť sa správnosťou zmeny. Keď je kód jasnejší je možné dokonca spozorovať dizajnovú stránku, ktorá mohla byť predtým ťažko alebo nemusela byť vôbec badateľná.

Objavenie chýb pri refaktoringu

Častokrát sa pri snahe prečítať a porozumieť kód stáva, že objavíte dosiaľ nepoznané chyby. Niektorí ľudia sa pozrú na úryvok kódu a hneď objavia nejakú chybu, ale nie každý ma nato talent. Avšak pri snahe o reštrukturalizáciu sa musíte hlbšie zamyslieť ako daný kus kódu pracuje, a vtedy môžete objaviť nejaké chyby a vymyslieť lepší spôsob, akým má daná časť fungovať. Objasnením štruktúry programu je možné vyjasniť predpoklady pre objavenie nových chýb v kóde.

Refaktoring napomáha rýchlejšiemu programovaniu

V konečnom dôsledku refaktoring pomáha k rýchlejšiemu vyvíjaniu softvéru. Vylepšuje dizajn, čitateľnosť, znižuje počet chýb a tým celkovo vylepšuje kvalitu softvéru. Dobrý dizajn je základ pre rýchle vyvíjanie aplikácií. Bez toho je možné dosiahnuť veľký progres v začiatkoch vývoja, ale v neskorších fázach bude zlý dizajn s určitosťou bremenom, ktoré bude celý proces spomaľovať. Veľa času potom zaberie hľadanie a opravovanie chýb, miesto pridávania novej funkcionality. Refaktoring umožňuje vyvíjať softvér rýchlejšie, pretože bráni rozpadávaniu sa dizajnu, ba dokonca pomáha jeho vylepšovaniu.

Kedy je potrebné robiť refaktoring

Podľa Fowlera nie je dobré si stanoviť určitý časový interval, ktorý sa bude opakovať a bude určený pre vykonávanie refaktoringu [6]. Refaktoring je potrebné robiť z času na čas v malom množstve a nie naraz. Keď napríklad chcete robiť chvíľu niečo iné ako bežne, môžete urobiť nejaké zmeny v kóde s cieľom refaktoringu. Alebo keď sa dostanete k časti kódu, ktorú považujete za nie úplne jasnú, vtedy sa môžete uchýliť k tejto činnosti.

Môžete sa riadiť aj pravidlom, ktoré sa často uplatňuje: Keď programujete prvý krát niečo, tak to urobíte. Druhýkrát keď robíte to isté, zastavíte sa nad tým, že to ste už robili, ale urobíte to duplicitne. Tretí krát ak idete robiť to isté, vtedy je správny čas na refaktoring.

Je potrebné testovanie?

Pravidelným dodržiavaním týchto postupov, ktoré sme popísali je možné dosiahnuť kvalitný, zrozumiteľný, čitateľný a udržiavateľný kód aj v robustných aplikáciách, avšak to všetko nám nedokáže zaručiť aj stopercentne správnu funkcionalitu. Máme kvalitný kód, tým pádom si povieme, že nepotrebujeme testovanie. Avšak podľa mňa sme na omyle. Z vlastných skúseností môžem povedať, že sa stáva v softvérových spoločnostiach, že vývojár niečo naprogramuje, sám si to aj vyskúša či mu to ide ako má a ak áno posunie to ďalej na testovanie. Avšak tu sa stáva, že až pri testovaní testerom sa odhalí nečakané správanie aplikácie, na ktoré by programátor neprišiel. Je to preto, lebo vývojár, keďže sa nevenuje testovaniu naplno, nevie odhadnúť všetky situácie, ktoré môžu nastať pri používaní tejto aplikácie. A ako človeka, ktorý stojí za vývojom danej aplikácie, by ho možno nenapadlo pracovať s tou aplikáciou spôsobom, akým sa nato pozerá bežný používateľ. Nejaká zmena v kóde môže ovplyvniť správanie úplne iného komponentu v inej časti aplikácie. Pri jednoduchom otestovaní funkcionality, ktorú programátor zmenil si to nemusí všimnúť a odhalí to až komplexné testovanie testerom. Takisto môže programátor svojim zásahom ovplyvniť aj funkcionalitu, s ktorou sa zaoberá niekto iný. Pri jednom školskom projekte sa nám s kolegom stalo, že sme mali naprogramovanú aplikáciu ako semestrálny projekt a mysleli sme si, že je vo finálnej fáze, kedy by už mala bez problémov fungovať. Avšak pri prezentovaní na cvičení túto aplikáciu začal testovať vedúci cvičenia, ktorý ako keby zmyslov zbavený klikal po celej aplikácii „rýchlosťou svetla“ a na miesta, kde by bežného používateľa ani nenapadlo kliknúť. Týmto spôsobom sme samozrejme zistili, že aplikácia nie je úplne stabilná a je tam ešte niekoľko chýb.

Avšak, niekedy ani testovanie neodhalí všetky chyby. V praxi sa stáva, že sa zákazníkovi odovzdá softvér, ktorý je považovaný za funkčný a zákazník potom hlási nejaké chyby. Dokonca sa raz stalo, že v istej inštitúcii bol nasadený nový softvér. Avšak táto inštitúcia hlásila jeho pád, tak sa vývojári snažili vylúskať oriešok, kde je problém. Keď sa im to po dlhom čase nedarilo, tak sa prišli pozrieť osobne k zákazníkovi a zistiť v akej situácii nastane pád aplikácie. Nakoniec prišli nato, že používateľka, ktorá pracovala s týmto softvérom písala a zadávala doňho údaje tak rýchlo, že ten ich nestačil spracovávať a nastal pád.

Záver

Kvalitný zdrojový kód neznamená hneď kvalitný softvér. Je to len časť z rôznych aspektov, od ktorých sa odvíja kvalita softvéru. Avšak napriek tomu je jej dôležitou súčasťou, ktorú je potrebné neustále vylepšovať. Na jeho dosiahnutí sa podieľajú v najväčšej miere vývojári. Môžu to dosiahnuť rôznymi spôsobmi napríklad aj tými, ktoré sú popísané v tejto eseji a to efektívne komentovanie kódu, metódy samo dokumentovania kódu, analyzovanie dlhodobého vplyvu zmien v kóde a používanie metód refaktoringu. Avšak napriek tomu na dosiahnutie celkovej kvality softvéru dodržiavanie týchto pravidiel samotné nestačí. Dôležitou súčasťou je fáza testovania. Kvalitný softvér sa podľa nás bez tejto fázy nezaobíde. Preto sa podľa nášho názoru nedá dosiahnuť kvalitný softvér bez fázy testovania. Tvorba softvéru je zložitý proces tvorený z viacerých fáz a kvalitu dosiahneme len súčinnosťou všetkých týchto fáz.

Použitá literatúra

1. Herzig, K.S.: Capturing the Long-Term Impact of Changes. In: *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 2 (2010).
2. Raskin J.: *Comments Are More Important Than Code.* 2005.
3. Gill, N. S.: *Factors Affecting Effective Software Quality Management Revisited*. New York, NY, 2005.
4. Knuth, D. E.: *Literate programming, 1992*
5. Stroggylos, K., Spinellis, D.: Refactoring – Does it improve software quality? *Int. Conference on Software Engineering*, 2007.
6. Fowler, M., Beck, K., Brant, J. , Opdyke, W., Roberts, D.: *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

Annotation

Is it possible to create quality software without testing?

We encounter the term quality every day in all areas of life. This term has strategic importance even in software engineering. However, achieving quality in software is a difficult process. Currently, there are many ways to do so. One of them is the improvement of source code. In the essay, we focus on the ways developers can improve the source code and thus improve software

quality. Sometimes we encounter the attitudes that the principles that enhance the source code, are a guarantee for quality software. Such a consideration can be misleading. The life cycle of software development involve several phases and one of them is testing. In the essay we deliver our opinion, whether it is possible to achieve good software quality without testing.