(R)EVOLÚCIA V TESTOVANÍ?

Testovať, testovať, testovať, aj keby sekery z neba padali.

Miloš Auder

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

Abstrakt. *Softvérové inžinierstvo ako relatívne mladé odvetvie prešlo za niekoľko desiatok rokov neskutočným vývojom. S rozvojom softvérového inžinierstva sa samozrejme menil aj spôsob, akým sa vyvíjaný softvér testuje. Testovanie softvéru sa s postupom času profesionalizovalo a automatizovalo, keďže sa vytvárané systémy stávali väčšími a zložitejšími a cena neskoro objavenej chyby rástla. Dnes máme na svete niekoľko prístupov k testovaniu softvérového produktu. V mojej eseji sa zamýšľam nad tým, kde sa testovanie nachádza teraz a kam bude tento vývoj ďalej smerovať. Spomínam niektoré známe techniky ako napríklad metóda čiernej a bielej skrinky, ale aj modernejší prístup ako testovaním riadený vývoj. V eseji uvažujem, kde je ktorú metódu vhodné použiť vzhľadom na veľkosť projektu a fázu vývoja, kto v tíme má testovať a ako. Na záver pripojím svoj názor na testovanie nášho projektu na predmete Tímový projekt.*

Kľúčové slová: *testovanie, biela a čierna skrinka, testovaním riadený vývoj*

Úvod

Nemusíte byť študovaným odborníkom, aby ste aspoň v podvedomí vedeli, čo to testovanie je. Veď predsa každý z nás deň čo deň niečo testuje, či už v domácnosti napríklad pri varení, majstrovaní v dielni alebo na nákupoch. No jednoducho takmer všade. Pri vývoji softvéru je to jedna zo základných aktivít, a je taká stará ako samotné písanie kódu. Keď už niečo dlho vytvárame, snažíme sa samozrejme aj o to, aby náš produkt bol na konci kvalitný a teda aby zákazníkovi vyhovoval po všetkých stránkach.

Práve na zabezpečenie kvality nám slúži testovanie, pretože nám poskytuje reálnu odozvu o správaní sa systému. Avšak postupom rokov, ako sa menili techniky programovania, analýzy, návrhu, používané technológie a mnoho iných faktorov súvisiacich s vývojom softvéru, museli sa meniť aj spôsoby ako zaistiť jeho kvalitu. Systémy sa stávali a stávajú robustnejšími a zložitejšími, sú na ne kladené rôzne požiadavky, či už na výkon, efektivitu, správnosť a podobne. Testovanie si skrátka nemohlo dovoliť zaspať dobu a muselo sa prispôsobovať a meniť, občas menej a občas viac radikálne.

Pravek testovania

Aby sme sa mohli pozrieť na súčasnosť a zamyslieť sa nad budúcnosťou testovania, domnievam sa, že je dôležité, ale určite aj zaujímavé sa pozrieť na jeho minulosť. Podľa [1] prvá literatúra venujúca sa testovaniu začala vychádzať v sedemdesiatych rokoch 20. storočia a prvá konferencia zasvätená testovaniu softvéru sa uskutočnila v roku 1972. Je nám však jasné, že testovanie tu už muselo byť aj predtým, že jednoducho nespadlo z jasného neba, ale dospelo sa k nemu ako k samostatnej problematike nejakým vývinom a postupným získavaním skúseností. Podľa [2] existovalo v minulosti niekoľko časových období, v ktorých sa na testovanie pozeralo rôzne. Ešte predtým ako sa však dalo vôbec hovoriť o nejakom testovaní softvéru, bolo testovanie zamerané najmä na hardvér, a teda problémy softvéru sa brali ako súčasť hardvérovej spoľahlivosti.

Prvým takýmto obdobím bolo tzv. obdobie orientované na debugovanie, kedy sa debugovanie a testovanie špeciálne nerozlišovalo. Prvé články na túto tému napísal Alan Turing (1949,1950), kde sa zamýšľal nad tým, ako vieme, že program už uspokojuje požiadavky naň dané.

Ďalšie obdobie sa začína v roku 1957, kedy Charles Baker rozlíšil medzi debugovaním a testovaním, kde debugovanie bolo prostriedok ako sa uistiť že program bude bežať, zatiaľ čo testovanie nás malo uistiť, že program vyrieši daný problém. Počas tohto obdobia debugovanie aj testovanie znamenalo nájsť, lokalizovať, identifikovať a opraviť chybu.

V roku 1979 Mayers popísal testovanie ako proces vykonávania programu so zámerom nájdenia chýb. Tento model sa nazýval deštručný, pretože jeho cieľom nebolo ukázať že program chyby neobsahuje (ako tomu bolo v predchádzajúcom období), ale naopak, že program chyby obsahuje. S týmto prístupom sa menili aj testovacie dáta tak, aby sme mali väčšiu pravdepodobnosť že pri testovaní chybu objavíme. A aj keď táto perioda trvala len krátko, podľa mňa je to zásadný zlom v tom, ako treba o testovaní premýšľať.

Neskôr v osemdesiatych rokoch sa všeobecne prijala metodológia, ktorá dáva testovaniu za úlohu hodnotiť produkt počas životného cyklu softvéru. Každá fáza životného cyklu má k sebe pridruženú istú množinu aktivít a produktov (V-model).

Doteraz bolo testovanie považované iba za akýsi prídavok k procesu vývoja [3]. Bolo prenechávané menej zručným, menej zarábajúcim ľuďom, pre ktorých testovanie bolo niečo ako vstupným bodom pre ich kariéru vo vývoji softvéru. Existovali taktiež prostriedky, ktoré zabezpečovali istú úroveň automatizácie, ale tieto nástroje boli drahé, zložito sa používali a boli neefektívne. Dôležité je taktiež povedať, že manažment v tomto

období bol zväčša oddelený od testovania softvéru, keďže predpokladali že je to niečo, čo sa proste spraví.

Kde sa testovanie nachádza dnes?

Miera profesionality testovania v posledných dvoch desaťročiach nesmierne narástla. Testovanie softvéru je momentálne jeden samostatný odbor. Vo firmách existujú testovacie oddelenia, tester ako pracovná pozícia už nie je ničím výnimočným. O testovaní vychádzajú publikácie, články, usporiadavajú sa konferencie a prijímajú sa rôzne štandardy. Toto všetko je síce veľmi pekné, ale aké zmeny to prinieslo do reálneho prostredia softvérového inžinierstva? Aké techniky sa dnes v testovaní používajú? Kto a čo všetko testuje?

Testovanie je akákoľvek aktivita zameraná na vyhodnocovanie schopnosti systému dávať požadované výsledky a riešenia [4]. Existuje niekoľko rozdelení podľa rôznych noriem, v tejto eseji uvediem dve základné rozdelenia. Podľa toho na akej úrovni softvér testujeme môžeme testovanie rozdeliť na:

- Testovanie jednotiek (unit testing) – ide o testovanie najmenšej jednotky systému
- Testovanie komponentov (component testing) – overovanie funkcionality jednotlivých komponentov
- Testovanie integrácie (integration testing) – testuje integráciu viacerých komponentov spájajúcich sa do väčšieho celku
- Testovanie systému – tu prebieha testovanie systému ako celku

Druhým rozdelením, na ktoré by som sa chcel bližšie pozrieť, je rozdelenie na metódy (techniky) ktorými vieme softvér testovať. Vo všeobecnosti sa hovorí o dvoch:

- Metóda čiernej skrinky (black-box testing) – testovacie dáta sú odvodené z funkcionálnych požiadaviek, pričom vnútorná štruktúra sa neberie do úvahy
- Metóda bielej skrinky (white-box testing) – tu naopak poznáme vnútornú štruktúru testovaného celku (poznáme jeho kód)

Myslím si, že tieto dve metódy sú natoľko známe, že hlbšie definície nie sú potrebné. Zamyslime sa radšej nad tým, vzhľadom na predchádzajúce rozdelenie podľa úrovní, akú techniku kde použiť. Zjednodušene by sa dalo povedať, že na každú úroveň vieme použiť obe techniky testovania. Avšak som toho názoru, že na testovanie jednotiek a komponentov je lepšie použiť metódu bielej skrinky, pretože keď poznáme kód, vieme testy navrhnúť tak, aby sme prešli a overili si správnosť a dostupnosť každej z možných ciest v danej jednotke. Takto si vieme preveriť, či je kód napísaný správne. Myslím si, že takéto testovanie by mal zväčša vykonávať samotný softvérový inžinier (programátor), pretože sám najlepšie pozná svoj kód a jeho fungovanie, a bol by určite najradšej, keby si mohol kód opraviť sám bez toho, aby ostatní vedeli aké chyby spravil. Avšak to, že pozná svoj kód najlepšie je aj najväčším rizikom neodhalenia chyby, pretože je príliš zaujatý a nevie sa na celý problém pozrieť s nadhľadom a nezaujate.

Testovanie metódou bielej skrinky je určite vhodné aj pre testovanie integrácie viacerých komponentov do celku. Testy sú písané tak, aby sa preskúmali rozhrania medzi

jednotlivými komponentami. Tieto testy samotné môžu byť aj na princípe metódy čiernej skrinky, pričom však tester musí chápať že pri vykonávaní tohto testu musia spolupracovať viaceré komponenty spolu. Takýto typ testovania by mal podľa mňa vykonávať tester, aby odbremenil od takéhoto testovania programátora.

Pre testovanie celého systému, alebo aj jeho jednotlivých komponentov sa nám hodí metóda čiernej skrinky. V takomto prípade by bolo príliš zložité naštudovať si celú vnútornú štruktúru systému. Vychádzame zo špecifikácie, vieme aké výstupy očakávať pri konkrétnych vstupoch. Článok [5] predstavuje päť možností testovania, založených na technike čiernej skrinky, ktoré sa hodia na testovanie objektovo-orientovaného systému a vychádzajú z UML notácie. Prvou z nich je testovanie založené na prípadoch použitia, kedy sú jednotlivé prípady použitia rozpísané, následne je tok udalostí prenesený do grafu a doplnený o výnimky. Pri tomto spôsobe sa neberú do úvahy algoritmy systému ani interakcia medzi jednotlivými komponentami. Druhou možnosťou je testovanie založené na kolaboračnom diagrame, ktorý nám vraví, ako medzi sebou objekty komunikujú. Treťou a štvrtou možnosťou je vychádzať z formálnej špecifikácie, tu nájdu uplatnenie prostriedky špecifikácie ako Object-Z alebo OCL. Piatou možnosťou je vychádzať pri príprave testov z rozšírených prípadov použitia, kedy vytvárame tzv. scenáre a kombinujeme ich s potrebnými triedami a ich parametrami. Autori odporúčajú kombinovať metódy testovania vychádzajúce z OCL a rozšírených prípadov použitia.

Samozrejme, keďže všetko na svete má svoje výhody a nevýhody, obe priblížené techniky nie sú výnimkami. Hlavnou výhodou techniky čiernej skrinky je, že testeri sa nemusia starať o vnútornú reprezentáciu aplikácie, a teda ľahšie sa im vytvárajú testy keďže im stačí prechádzať aplikáciou ako keby boli koncovými používateľmi. Nemusia sa zaoberať rôznymi cestami, ktorými môže kód viesť, stačí im preskúmať všetky možnosti ktoré poskytuje GUI testovaného systému. Sústreďujú sa len na validné/nevalidné vstupy a kontrolujú či dostanú správne výsledky. Z tohto vyplývajú aj nevýhody, keďže ak sa nám zmení prostredie pre vstup, napríklad sa zmení používateľské rozhranie, môže sa nám tým rozhádzať celý test. V tomto prípade by sa dalo ironicky podotknúť, že je škoda, že metóda čiernej skrinky nie je viac podobná metóde bielej skrinky. Dostávam sa teda k najväčšej výhode metódy bielej skrinky a tou je schopnosť pozrieť sa do „vnútra programu“. Toto nám môže zaistiť istú celistvosť testovacích skriptov pri často sa meniacom GUI, za predpokladu že sa nemenia napríklad mená premenných a objektov. V situáciách, kde je kritické, aby sme otestovali každú možnú cestu ktorou môže program viesť, je použitie techniky bielej skrinky jedinou možnou voľbou. Na druhú stranu, pri zložitosti dnešných systémov si myslím, že musí byť ťažké pre testera aby poznal programátorskú stránku každej časti systému, preto je v týchto prípadoch metóda bielej skrinky takmer nepoužiteľná, alebo si vyžaduje naozaj zručného a skúseného testera. Výhody a nevýhody sú teda jasné, dôležité je správne sa rozhodnúť pri práci na projekte, v akej fáze, na akej úrovni a do akej miery treba jednotlivé techniky použiť. Pred začatím práce by sa mali vývojári spýtať sami seba niekoľko otázok, ako napríklad kto bude koncovým používateľom, bude sa často meniť GUI, a ak áno, bude tým veľmi ovplyvnený kód aplikácie? V akom jazyku bude aplikácia písaná, kde bude nasadená? Ak by napríklad išlo o webovú aplikáciu, kde by používatelia zadávali informácie cez sériu formulárov, vhodné by podľa mňa bolo použiť metódu čiernej skrinky. V iných typoch aplikácií, kde sa niečo ráta, môže tam nastať mnoho vetvení, alebo treba otestovať extrémne situácie, sa

skôr hodí metóda bielej skrinky. Čo sa týka testovania berúc do úvahy veľkost projektu, nerozlišoval by som striktne medzi použitím metódy bielej a čiernej skrinky. Domnievam sa, že každá nájde svoje uplatnenie či už pri malých, stredne veľkých, alebo veľkých projektoch. Treba si správne zhodnotiť, aké máme k dispozícií finančné prostriedky, ľudské zdroje a koľko času nám ostáva.

Nesporne existuje veľa ďalších spôsobov testovania, ktorých cieľom je okrem iného testovať splnenie nefunkcionálnych požiadaviek na systém. Nie je cieľom tejto eseje ich podrobne rozoberať, ale spomenul by som aspoň že existujú. Patria medzi ne záťažové testovanie, testovanie spoľahlivosti a v poslednej dobe je veľmi aktuálna otázka bezpečnosti vyvýjaných systémov, a teda testovanie bezpečnosti.

Testovanie softvéru môže byť drahou záležitosťou. Jedným zo spôsobov ako skrátiť čas a cenu testovania je jeho automatizácia [4]. Nástroje na automatizáciu testovania však nie sú zatiaľ, žiaľ, všeobecne použiteľné a rozšíriteľné. Pri testovaní je stále potrebné značné množstvo ľudskej práce. Automatizácia testovania, ako významný krok v testovaní softvéru, úzko súvisí s testovaním riadeným vývojom, ktorému sa venujem v ďalšej kapitole.

Testovaním riadený vývoj

Podľa [6], testovaním riadený vývoj nie je žiadnou novinkou a sporadicky sa používal už po desaťročia. Najstaršia známka o jeho použití je zo šesťdesiatych rokov minulého storočia z projektu Mercury v NASA. Testovaním riadený vývoj si získal pozornosť v poslednej dobe ako jedna zo súčastí extrémneho programovania a agilného prístupu k vývoju softvéru.

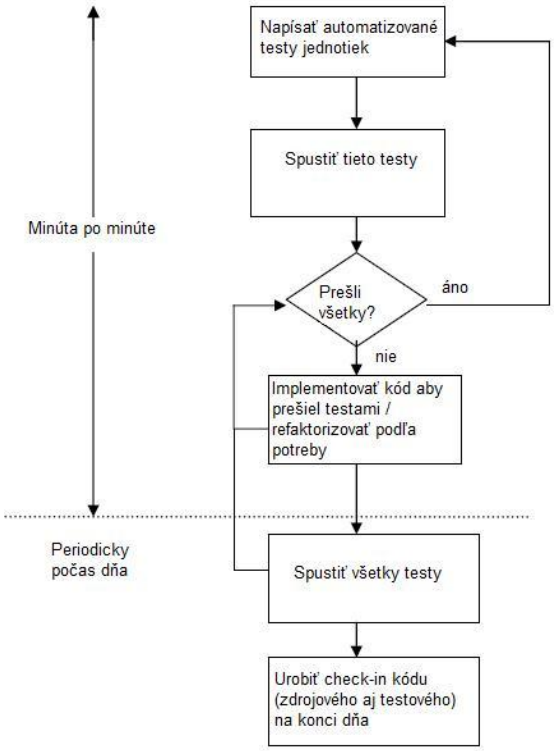
O čo teda ide? Čo je na testovaním riadenom vývoji také špeciálne? Všetky techniky a spôsoby, ktoré som doteraz spomenul, mali jednu spoločnú črtu. Príprava testov a a samotné testovanie sa uskutočňovali až *po* vytvorení nejakej časti systému, alebo v tom horšom prípade, až na koniec po jeho dokončení. Často krát však na toto nemusí ostať čas, alebo ho ostane len veľmi málo. Následné objavovanie a opravovanie chýb môže byť vcelku drahou a nepríjemnou záležitosťou pre vývojárov.

Pri vývoji riadenom testovaním sa zavádza príprava testov *pred* písaním kódu. Na obr. 1 vidno grafické znázornenie postupu pri praktizovaní vývoja riadeného testovaním. Má niekoľko špecifík a výhod, s ktorými však ruka v ruke prichádzajú aj nevýhody.

Testovaním riadený vývoj má niekoľko pozitív, ale tou hlavnou asi je, že núti vývojára ísť po malých krokoch. Je určite ľahšie nájsť a opraviť chyby po pridaní iba pár riadkov kódu, ako keď ich hľadáme po pridaní niekoľko stovák alebo tisícov riadkov. Príprava testov je v tomto prípade nie len vecou overovania správnosti implementovaných častí, ale taktiež záležitosťou návrhovou. Na nízkej úrovni si vieme lepšie navrhnúť jednotlivé triedy, metódy, rozhrania a podobne. Tým, že píšeme testy a silno sa zamýšľame nad vytváraním kódu ešte pred jeho písaním, vedia nám tieto testy slúžiť do istej miery aj ako dokumentácia k vytváraným častiam. Jednou z ďalších výhod je určite aj to, že kód je písaný tak, aby bol okamžite testovateľný, čo do určitej miery šetrí čas. S týmto argumentom je možné nesúhlasiť, keďže niekto by povedal, že písanie toľkých testov nám v skutočnosti čas strávený na projekte predĺži. Áno, je to pravda, ale som toho názoru, že ak je našim prvoradým zámerom vyrobiť produkt kvalitný, trocha času na viac

nám nemôže robiť vrásky na tvári. Lenže ako sa vraví, čas sú peniaze, preto si už musí každý vývojový tím pred začatím práce dobre premyslieť a zvážiť, aký pomer zvoliť medzi atribútmi kvalita/čas (peniaze).

Týmto som spomenul zrejme najväčšiu nevýhodu testovaním riadeného vývoja, a teda predlžovanie času stráveného prípravou testov. Tento prístup však môže mať aj iné nedostatky. Pri takomto spôsobe vývoja nám písanie testov môže dať akýsi falošný pocit istoty, že iné testy už nie sú potrebné. Testovanie jednotiek ako také nám však určite samé nevystačí, možno len pri naozaj malých aplikáciách. Treba jednoznačne používať aj iné testovania, také, ktoré idú za rámec funkčnosti nejakej jednotky. No a v neposlednom rade vždy môže zlyhať ľudský faktor. Predsa nie všetci programátori musia byť zbehlí a šikovní v písaní testov. Riešením môže byť zaškolenie, alebo spárovanie s kolegom, ktorý je v písaní testov zručnejší. Nuž a nakoniec nie každému sa môže takýto prístup k vývoju páčiť. Môžeme sa úplne spoľahnúť, že každý programátor bude testovaním riadený vývoj svedomite praktikovať? Ak nie, mali by sme členov tímu, ktorí tak nerobia, nejakým spôsobom motivovať, alebo po prípade vyhodiť z tímu. V najhoršom možnom prípade môže byť nutné, aby tím jednoducho od testovaním riadeného vývoja upustil.



Obr. 1. Testovaním riadený vývoj [6].

Quo vadis, testovanie?

Výskumníci a vývojári neustále hľadajú nové spôsoby ako spraviť testovanie lepším, ako na ňom ušetriť čas a finančné prostriedky, za predokladu, že ostane zachovaná vysoká kvalita. Autorka v článku [1] vyzdvihuje štyri najväčšie výzvy v testovaní. Sú nimi:

1. Univerzálna teória testovania
2. Testovanie založené na modelovaní – modelovanie založené na testovaní
3. 100 % automatizované testovanie
4. Maximálne efektívne testovacie inžinierstvo

Jedným z najdlhších snov pri testovaní softvéru je vymyslieť jeden súdržný a precízny nástroj, na ktorý by sa mohli testeri odvolávať pri výbere techniky testovania. Určite by bolo príjemné mať k dispozícii aparát, ktorý by združoval cieľ testovania s výberom najadekvátnejšej techniky a vedel by sa sám adaptovať na prostredie. Myslím si, že z uvedených snov v testovaní sa jedná o ten najmenej reálny, avšak výskum rýchlo napreduje, nechajme sa teda prekvapiť.

Hlavnou myšlienkou testovania založeného na modelovaní je použiť modely vytvorené pri návrhu softvéru a riadiť sa nimi pri testovacom procese, konkrétne povedané, automaticky generovať testy. Teda ide o to použiť moderný modelovací nástroj, napríklad UML, a adaptovať naň testovacie techniky tak efektívne, ako to len pôjde. Užitočný by bol aj opačný proces, a teda modelovanie založené na testovaní. Ten nás núti uvažovať ako by sme mali ideálne navrhnúť softvér tak, aby bol čo najlepšie testovateľný. Osobne sa mi táto idea veľmi páči, vo veľkej miere by uľahčila testovania v zmysle toho, že by sme vedeli, že sme naozaj všetko otestovali (ak to bolo v návrhu). Splnenie tejto výzvy sa mi javí ako veľmi reálne, technicky uskutočniteľné, a ak už nie sú na trhu prostriedky, ktoré toto aspoň čiastočne zavádzajú, som presvedčený, že do pár rokov sa určite objavia.

Veľká časť výskumu sa v týchto časoch sústreďuje na zlepšenie stupňa automatickosti testovania. Či už je to zlepšovaním techník na automatické generovanie testovacích vstupov, alebo nachádzaním inovatívnych prístupov k podpore automatizácie testovacieho procesu. Cieľom je teda výkonné testovacie prostredie, kde by sa automaticky generovali vhodné testy, vykonávali by sa a my by sme dostali už len výslednú správu. Pokroky v tejto oblasti nastali hlavne pri testovaní jednotiek, ale ani tu automatizácia stále nie je na dostatočnom stupni. Som toho názoru, že stupeň automatizácie testovania bude aj naďalej rásť, avšak nemyslím si, že niekedy dosiahneme 100%.

Maximálne efektívne testovacie inžinierstvo autorka článku [1] vyhlasuje za najvyšší cieľ výskumu testovania softvéru. Podstata je pri tom veľmi jednoduchá, a to urobiť testovací proces čo najúčinnejším za čo najmenšiu cenu. Povedal by som, že všetky výzvy spomenuté vyššie, sú len akousi podmnožinou tejto výzvy. To, čo neustále podnecuje vývojárov pracovať na stále lepších metódach a technikách testovania, je neustále narastajúca zložitosť vyvýjaných systémov. Táto zložitosť neovplyvňuje len systém, ale aj prostredie do ktorého je takýto systém nasadený. Je ťažko povedať, či je vôbec možné splniť túto veľkú výzvu? Myslím si, že nie, a tak by to aj malo byť. Veď predsa vždy je čo zlepšovať.

Testovanie tímového projektu

Pri testovaní nášho projektu je treba v prvom rade vychádzať z témy a povahy projektu. V našom prípade pôjde o heuristický evolučný pravidlami riadený umelý život. Povaha projektu je skôr výskumná, zaujíma nás to, čo zistíme pri pozorovaní umelého života. Aplikácia sama o sebe bude veľmi jednoduchá, pozostávajúca z dvoch obrazoviek, kedy na jednej používateľ zadá parametre sveta a spustí simuláciu. Druhá obrazovka bude vizualizačná, kde si budeme môcť simuláciu pozrieť, poprípade do nej zasahovať. Dôležité bude zabezpečiť to, aby sa naše nastavenia bez chýb preniesli do následných výpočtov a grafickej simulácie. Testovanie v tomto prípade budeme robiť vždy po pridaní nového parametra sveta, kde sa uplatní metóda bielej skrinky. Pri vizualizácii sa hodí metóda čiernej skrinky, kde si budeme chcieť skontrolovať, či to, čo sme si na začiatku zadali, sa udeje aj v simulácii. Testovanie bude prebiehať manuálne a testovať bude viac či menej každý člen tímu. Keďže výsledky odpozorované zo simulácií sú pre náš projekt kľúčové, niet tu priestoru pre chybu.

Záver

Mať kvalitný produkt by malo byť pre výrobcov všeobecne prioritou. Vo svete softvéru to platí dvojnásobne, a keďže testovanie je prostriedok, cez ktorý je možné vysokú kvalitu dosiahnuť, nemalo by sa odsúvať na vedľajšiu koľaj, ale naopak, mala by sa mu venovať zvýšená pozornosť. Vývojárom je jasné, že stále treba hľadať nové a lepšie spôsoby ako testovať. V mojej eseji spomínam začiatky testovania, aké techniky sa bežne používajú dnes, a kde a kedy je vhodné ich použiť. Testovanie je dnes už aj návrhovou záležitosťou, čoho dôkazom je testovaním riadený vývoj. Ku koncu sa snažím nazrieť trocha do budúcnosti testovania softvéru. Výzvy, ktoré si výskumníci a vývojári pred seba postavili sú naozaj veľké, bude zaujímavé sledovať, ktoré, a do akej miery sa im podarí zdolať. Na koniec eseje sa zamýšľam nad spôsobom testovania nášho tímového projektu. Som sám zvedavý, ako bezchybne náš projekt spravíme, a či sa nám podarí objaviť všetky chyby.

Použitá literatúra

1. Bertolino, A.: Software Testing Research: Achievements, Challenges, Dreams. In *2007 Future of Software Engineering* (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC (2007), 85-103.
2. Gelperin, D. and Hetzel, B. 1988. The growth of software testing. *Commun. ACM* 31, 6 (Jun. 1988), 687-695.
3. Nguyen, Q. H., Pirozzi, R.: The early revolution of software testing. Software test professionals conference, San Mateo, California (2007)
4. Pan J.: Software testing. Carnegie Mellon University (1999)
5. Kwang Ik S.; Eun Man Ch.: Comparison of Five Black-box Testing Methods for Object-Oriented Software, *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on* , vol., no., pp.213-220, 9-11 (2006)

6. Bhat, T. and Nagappan, N.: Evaluating the efficacy of test-driven development: industrial case studies. In *Proceedings of the 2006 ACM/IEEE international Symposium on Empirical Software Engineering* (Rio de Janeiro, Brazil, September 21 - 22, 2006). ISESE '06. ACM, New York, NY (2006), 356-363.

Annotation

(R)evolution in testing?

Software engineering as a relatively young branch has gone through an extraordinary evolution during the last couple of decades. The method of testing the software changed with the development of software engineering. Software testing gradually became more professional and automatic, as the systems in development were becoming more extensive and difficult and errors found late became more expensive. Nowadays, there are several approaches to software testing. In this essay, I am contemplating the current state of the testing and its future tendencies. Some well-known methods are mentioned, such as black-box and white-box testing, but also more modern approach including test driven development. It is also discussed when to use which method according to the project and phase of the development, which person from the team should be testing and how. I will conclude with my opinion on testing our own project for the subject Team project.