Fakulta elektrotechnická
Katedra ekonomiky, manažerství a humanitních věd

Metódy overovania kvality softvérových aplikácií a náklady na kvalitu

BAKALÁRSKA PRÁCA

Studijní obor: Manažerská informatika

Vedúci práce: Ing. Josef Černohous

Viktória Strharská

Praha 2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Strharská** Jméno: **Viktória** Osobní číslo: **422966**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra ekonomiky, manažerství a humanitních věd**

Studijní program: **Softwarové technologie a management**

Studijní obor: **Manažerská informatika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Metody oveřování kvality softwarových aplikací a náklady na kvalitu

Název bakalářské práce anglicky:

Quality verification methods and cost of quality

Pokyny pro vypracování:

1. Metody ověřování kvality softwarových aplikací
2. Měření kvality softwaru
3. Aspekty ovlyvňující náklady na kvalitu softwaru
4. Rozbor a vyhodnocení měření kvality softwaru z reálných IT projektů

Seznam doporučené literatury:

1. Jeffrey Rubin, Dana Chisnell: Handbook od Usability Testing, Wiley, 2008, 978-0-470-18548-3
2. Petr Roudenský, Anna Havlíčková: Řízení kvality softwaru - COMPUTER PRESS, 2013,9788025138168

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Josef Černohous, K13116 308b

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.03.2017** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: _____

_____	_____	_____
Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

_____ .	_____
Datum převzetí zadání	Podpis studentky

Prehlasujem, že bakalársku prácu na tému Metódy overovania kvality softvérových aplikácií a náklady na kvalitu som vypracovala samostatne, a že som uviedla všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe dňa ..... Podpis .....

Poďakovanie

Na tomto mieste by som chcela poďakovať v prvom rade vedúcemu práce pánovi Ing. Josefovi Černohousovi za odborné vedenie pri spracovaní mojej bakalárskej práce a za konzultácie k jej obsahu. Tiež by som chcela poďakovať svojmu blízkemu okoliu, ktoré mi ochotne poskytlo potrebné informácie a dostatok priestoru a podpory pre jej tvorbu.

Anotácia

Cieľom tejto bakalárskej práce je charakteristika základných metód zabezpečenia kvality softvérových aplikácií, metrík slúžiacich na meranie kvality na základe rôznych aspektov dôležitých pre konkrétny projekt a popis nákladov, ktoré sú spojené s odstraňovaním nedostatkov alebo zabezpečením vyššej kvality produktu. Práca taktiež približuje čitateľovi, čo pojem kvalita znamená a čo presne si pod týmto pojmom môže predstaviť

V praktickej časti je na konkrétnom projekte vysvetlené využitie jednotlivých metrík a metodík v praxi a porovnanie aktuálnej situácie so situáciu, kedy by testovanie softvéru neprebehlo, z čoho sú následne vyvodené dôsledky. Záver praktickej časti pre porovnanie využiteľnosti metodík popisuje v krátkosti priebeh testovania ďalšieho IT projektu.

Kľúčové slová

Kvalita, testovanie, náklady na kvalitu, validácia, verifikácia

Annotation

The aim of this bachelor thesis is to characterize the basic methods of quality assurance of software applications, quality measurement metrics based on various aspects relevant to a particular project and the description of costs that are associated with eliminating deficiencies or ensuring higher quality of the product. The work also brings to the reader what the term Quality means and what exactly can one imagines.

In the practical part, the concrete project explains the use of individual metrics and methodologies in practice and compares the current situation where the software testing did not proceed, and the consequences are drawn. The conclusion of the practical part of the comparison (of the applicability of the methodologies) briefly describes the course of testing of another IT project.

Keywords

Quality, testing, quality costs, validation, verification

Obsah

Úvod

Predmetom bakalárskej práce sú metódy overovania kvality SW aplikácií. Vo svojej práci sa snažím priblížiť, aké metódy overovania kvality sa v súčasnosti používajú a aké sú správne kritéria pre výber dostatočne flexibilných a efektívnych metód. Hneď na začiatku som sa zamerala na samotný pojem „kvalita“. Čo to presne kvalita je, sa len veľmi ťažko posúdi objektívne. Predsa len však existuje mnoho teórií a charakteristík.

Softvérové aplikácie (SW) majú v súčasnosti veľký vplyv na obchodný výsledok firiem. V konkurenčnom boji sú spoločnosti tlačené do čo najvyššej miery finančnej efektivity a flexibility zároveň. Tieto dva aspekty sa navzájom ovplyvňujú, ale častokrát sú aj v protiklade. Flexibilita vyžaduje od vývoja SW aplikácii čo najrýchlejšie dodanie, finančná efektivita ale požaduje primeraný pomer nákladov a výnosov.

Vývoj SW aplikácií je tak pod tlakom dodať riešenie čo najrýchlejšie, a za čo najnižšiu cenu. Jedným z dopadov je potom možný dopad na kvalitu vývoja a dodávky SW aplikácií. Tá musí byť ako flexibilná, tak aj finančne efektívna.

Aby sme dosiahli určitý stupeň kvality, existujú metódy, ktoré nám túto kvalitu pomôžu overiť a dosiahnuť tak čo najlepší výsledok. Nie každá z metód sa v reálnych IT projektoch využíva. Je to hlavne na rozhodnutí projektového manažéra, ktorý tak učiní na základe dostatočných informácií o projekte. V poslednej časti bakalárskej práce je popísaná konkrétna aplikácia metód zaistenia kvality v reálnom projekte. Tu je možné pozorovať ako konkrétne sa jednotlivé metódy používajú a aký majú prínos pre projekt.

V záverečnej časti popisu prvého projektu sú pre porovnanie podrobnejšie vysvetlené financie spojené s testovaním a náklady, ktoré by vznikli, ak by bolo testovanie úplne vynechané. Tu je zreteľne vidieť výhody, ktoré testovanie prináša aj po finančnej stránke. Nie je teda len zárukou dosiahnutia kvalitnejšieho produktu, ale tiež predpokladom pre zníženie nákladov na opravy vzniknutých chýb.

V poslednej časti bakalárskej práce je popísané použitie metód v projekte úplne iného charakteru, čo poukazuje na značné rozdiely v aplikácii jednotlivých metód vzhľadom na potreby projektu.

1. Rešerš existujúcich metód testovania

Prvá kapitola predstavuje štandardné metodiky, ktoré sa používajú pri zaisťovaní kvality softvérových aplikácií. Každá z metodík sa používa v inej fázy projektu a je vykonávaná rôznymi špecialistami od samotných vývojárov až po testovanie bežnými užívateľmi. Nie je nutné aplikovať v rámci jedného projektu všetky metodiky, avšak je dôležité si uvedomiť aké pri vytváraní softvéru hrozia riziká a na základe toho sa správne rozhodnúť akým spôsobom bude kvalita zaisťovaná. [1][2][6][8][16]

1.1 Testovanie v priebehu vývoja (Developer Testing)

Toto testovanie vykonáva vývojár v priebehu fázy výstavby celého životného cyklu vývoja softvéru. Má za cieľ odstrániť konštrukčné chyby pred tým ako je kód predvedený pred QA (quality assurance) [8]. Táto stratégia je určená pre zvýšenie kvality výsledného softvéru tak, aby bola pred QA predstavená v čo najvyššej kvalite. Tejto metódy testovania sa nezúčastňuje žiaden tester. Testy vývojár vykonáva vo forme integračných testov, jednotkových testov a niekedy aj systémových testov.

Tieto testy začínajú definovaním politík, ktoré vyjadrujú očakávania organizácie, čo sa týka spoľahlivosti, bezpečnosti, výkonu a dodržiavania predpisov. Potom, ako je tím oboznámený s týmito požiadavkami, sú implementované praktiky vývojárskeho testovania v súlade s danými požiadavkami.

Je dôležité podotknúť, že väčšinou kód netestuje programátor, ktorý ho vytvoril, ale testuje ho niekto iný. Táto časť testovania softvéru je najmenej nákladná. Je dôležité sa vyhnúť situácii, kedy je program bez kontroly programátora predaný testerom a tí zistia, že kód nie je spustiteľný. Tester sa tak úplne zbytočne pripravoval na testy a študoval dokumentáciu.

Avšak vynára sa otázka, či je to naozaj tak, či skutočne môžeme túto časť testovania považovať za najmenej nákladnú. Napriek tvrdeniam v literatúre, treba podotknúť, že práca vývojára stojí firmu viac peňazí ako práca testera, a to, že vývojár strávi určitý čas testovaním znamená, že za peniaze, ktoré ma určené na vývoj testuje, a to nie vždy musí byť lacnejšie ako v prípade, že by túto prácu vykonal tester.

1.2 Jednotkové testy (Unit Testing)

Tieto testy sú často krát považované za zbytočné z toho dôvodu, že na nich „nie je čas“. Napriek tomu však majú veľký potenciál aplikáciu pripraviť kvalitnejšie, vyhnúť sa chybám a pomôcť vytvoriť aplikáciu odolnejšiu. Toto testovanie je automatické a overuje teda korektnosť celého systému. Ako už napovedá slovo „unit“ ide konkrétne o testovanie správnej funkčnosti jednotlivých častí zdrojového kódu. Jednotkou je samostatná testovateľná časť, napríklad trieda či konkrétna metóda. Testovaná jednotka musí byť plne nezávislá na ostatných častiach kódu, preto je niekedy potreba vytvoriť pomocné objekty s pomocou ktorých, je daná časť kódu testovateľná. Pri testovaní sa sleduje chovanie jednotky a to, či dáta vyhovujú definíciám. Tieto testy väčšinou vykonáva vývojár. Testy síce vývoj zo začiatku trochu spomalia, ale zabezpečia spoľahlivejšiu aplikáciu. Nemôžu však zabezpečiť stopercentnú bezchybnosť. [3]

1.3 Integračné testy (Integration Testing)

Ide o overenie správnej integrácie jednotiek systému. Testuje sa prepojenie všetkých častí navrhnutého systému. Pri týchto testoch nás hlavne zaujíma ako funguje celok, preto môžeme používať aj iné triedy alebo databázu. Ide o začlenenie aplikácie do existujúceho prostredia. Integračné testy sú veľmi dôležité pri vývoji softvéru, pretože sa tu už pristupuje k tzv. unit case. Testovania sa zúčastňujú testeri, ktorí vykonávajú ako white-box[1] testy tak aj black-box[2] testy. Títo testeri sú rozdelení na dva tímy, a každý z nich pracuje samostatne.

1.4 Systémové testy (System Testing)

Po tom, ako je overená integrácia, prichádza na rad systémové testovanie. Aplikácia je overovaná z pohľadu zákazníka. Pripravia sa scenáre aké rôzne situácie môžu nastať a tie sa aplikujú na vytvorenú aplikáciu. Ak sa nájdu chyby poznamenajú sa, testy sa ukončia a chyby sa opravia. Potom sa testovanie opakuje, až kým nie sú odstránené všetky chyby. Súčasťou tejto úrovne sú funkčné aj nefunkčné testy. Systémové testy slúžia ako výstupná kontrola softvéru. Systémové testy sa vykonávajú vždy, i keď žiadna iná testovacia metóda pred tým nebola použitá.

[1] "White-box testy" – tester má prístup k obsahu programu a vie ako program pracuje
[2] "Black-box testy" – tester vie čo má testovaný software robiť, no nevie ako pracuje vo vnútri

1.5 Akceptačné testy (Acceptance Testing)

V tejto fáze je potrebné pripraviť systém na reálne nasadenie, a preto prebieha až po odstránení nedostatkov z predchádzajúcich testov. Testovanie je vykonávané buď samotným zákazníkom, koncovým užívateľom (BETA testy) alebo internými testermi. Scenáre k výkonu akceptačných testov pripravuje zákazník spoločne s dodávateľom. Testy môžu prebiehať buď ako užívateľské akceptačné testovanie (user acceptance testing), kedy užívateľ overuje vhodnosť systému pre použitie, prevádzkové testovanie (operational testing), kedy systémoví administrátori overujú údržbu systému, zálohu alebo obnovu systému alebo testovanie internými testermi. Nájdené nezrovnalosti medzi aplikáciou a špecifikáciou sú vrátené vývojárskemu tímu, a po oprave sú nasadené na prostredí u zákazníka. Zákazník väčšinou chyby o čakáva a preto je vhodné dohodnúť postup, ktorým budú chyby reportované vývojárskemu tímu a následne v čo najkratšej dobe opravené. Z tohto hľadiska delíme akceptačné testy na formálne (interní testeri s vopred určeným testovacím plánom), neformálne (testovanie nie je plánované, tester si volí spôsob testovania podľa vlastného uváženia) a BETA testy (bez interných testerov). BETA testy sú považované za najefektívnejšie a je pri nich odhalených veľa skrytých chýb. Príliš veľké zdržanie v tejto úrovni testovania môže mať fatálne následky na úspech celého projektu. Je vhodné si dopredu určiť akceptačné kritériá, alebo inak koľko defektov a akého rozsahu sa môžu pri akceptačných testoch objaviť.

1.6 Nezávislé testovanie (Independent Testing)

Tohto testovania sa zúčastňujú nezávislí testeri, tj. tí, ktorí nie sú súčasťou vývojového tímu. Testujú formou black-box testov, vďaka čomu prinesú do testovania o niečo viac užívateľský prístup ako interní testeri. Cieľom tohto testovania je priniesť nový pohľad na vec. Alternatívnym pohľadom na nezávislé testy je Independent Stakeholder Testing, ktoré je zamerané na potreby a záujmy všetkých, ktorých sa projekt týka (nie len pohľad užívateľa, ale i ďalších záujmových skupín).

2. Charakteristika softvérovej kvality

Keďže sa softvér stáva čím ďalej tým viac neoddeliteľnou súčasťou nášho života, zvyšujú sa tiež požiadavky na jeho kvalitu. Čo však presne slovo kvalita znamená? Z rôznych uhlov pohľadu má tento pojem rôzny význam. To nám názorne ilustruje práca spracovaná Davidom Garvinom. Ten predstavuje päť rôznych pohľadov na kvalitu [5]:

- **Transcendentálny pohľad:**

Častokrát sa nám stáva, že kvalitu hodnotíme na základe pocitov. Vnímame, že je niečo kvalitné, napriek tomu však svoj úsudok nedokážeme slovne zhodnotiť. Znamená to, že kvalitu je veľmi jednoduché rozpoznať, no neľahké definovať.

- **Užívateľský pohľad:**

Produkt je pre užívateľa kvalitný, ak spĺňa jeho požiadavky a očakávania. Je teda vhodný pre zamýšľané použitie.

- **Pohľad výroby:**

Úroveň kvality je určená mierou rozsahu, v akom produkt spĺňa špecifikácie.

- **Pohľad produktu:**

Kvalita sa viaže na vnútorné kvality, ktoré určujú kvality vonkajšie.

- **Pohľad ceny:**

Kvalita závisí na tom, koľko je za ňu zákazník ochotný zaplatiť

Snaha kvalitu definovať viedla k vzniku rôznych modelov kvality v rámci medzinárodných noriem. No keďže tieto normy stále trpia vážnymi nedostatkami sú postupne nahrádzané jednotným systémom noriem ISO/IEC 25000-25099.

Veľmi známym modelom kvality je model FURPS [1][9][10]. Tento model vytvorila spoločnosť Hewlett-Packard. Názov tohto modelu je zložený z prvých písmen piatich atribútov kvality dodaného systému.

Obrázok 1Model FURPS

Zdroj: https://qualidadebr.wordpress.com/2008/07/10/furps/

Charakteristika atribútov kvality na základe modelu FURPS:

- **Funkčnosť:**

Zameriava sa na hlavné funkčnosti a schopnosti programu, či softvér napĺňa požiadavky biznisu a podporuje biznis procesy.

- **Vhodnosť:**

Hodnotí sa hlavne z pohľadu koncového užívateľa, ako ľahko sa dá aplikácia použiť, akým celkovým dojmom pôsobí aplikácia, dokumentácia a školiace materiály.

- **Spoľahlivosť:**

Jedná sa o hodnotenie počtu a závažnosti chýb, presnosti spracovania vstupov a výstupov. Pre vyjadrenie spoľahlivosti sa často používa metrika MTBF (mean time between failures), čo je stredná doba medzi chybami alebo zlyhaniami. V tejto oblasti sa tiež sledujú možnosti obnovenia prevádzky a zotavenia sa z výpadku. Patria sem tiež záťažové testy a testy zotavenia aplikácie po zlyhaní niektorých komponent riešení.

- **Výkon:**

Hodnotenie celkovej rýchlosti odoziev systému a spracovanie kľúčových biznis aktivít. Zároveň sa sledujú aj technické parametre testovaného systému, napríklad vyťaženie zdrojov OS, zaťaženie sieťovej prevádzky, rozloženie záťaže na jednotlivé komponenty systému.

- **Udržateľnosť:**

Ďalším hľadiskom hodnotenia je oblasť údržby a podpory aplikácie, jej testovateľnosti. V tejto oblasti sa tiež hodnotí aj prispôsobiteľnosť a rozšíriteľnosť o nové

vlastnosti. Dôležitá je tiež schopnosť zapojenia aplikácie do existujúcich procesov podpory a údržby SW.

V poslednej dobe sa metóda FURPS rozšírila o znamienko +, na FURPS+ [9]. Postupom času totiž prestali jednotlivé kategórie dostačovať, a preto bolo zavedené toto „plus“, ktoré označuje, že je metóda rozšírená o ďalšie čiastkové elementy alebo podoblasti:

- **Implementácia:**

Rôzne obmedzenia, jazykové mutácie, špeciálne nástroje a HW

- **Rozhranie:**

Efektivita rozhrania na externé IT systémy

- **Operačné systémy:**

Špeciálne požiadavky na OS a jeho prípadnú úpravu, vrátane návrhu ďalšej správy

- **Obchodné a právne aspekty:**

Licencovanie a prípadné právne obmedzenia pre rôzne demografické regióny

3. Meranie kvality softvéru

Kvalitu meriame počas celého životného cyklu softvéru na procesoch, ktoré vo výsledku silne ovplyvňujú kvalitu samotného produktu. Meraním kvality jednotlivých procesov sa zaisťuje kvalita celého priebehu, čím sa primárne snaží predchádzať vzniku defektov. Medzi hlavné aktivity zaisťovania kvality softvéru patria definície, zavedenie procesov vrátane používaných noriem, procedúr, metrík, či nástrojov a následná kontrola ich dodržiavania. Riadenie kvality softvéru sa zameriava na výstupy z jednotlivých procesov, u ktorých overuje, či odpovedajú špecifikáciám a požiadavkám. To, či software odpovedá špecifikáciám, vykonáva presne to, čo od neho užívateľ vyžaduje overujú verifikačné a validačné aktivity.

[1][2]

3.1 Verifikácia

Verifikáciou overujeme, či čiastočný produkt odráža špecifikované požiadavky. Verifikácia požiadaviek zisťuje, či sú kompletné, konzistentné, vykonateľné a testovateľné. Na základe týchto požiadaviek vzniká návrh, ktorý musí byť správny a úplný. Takto vypracovanému návrhu odpovedá zdrojový kód. Ten opäť musí byť správny a testovateľný, umožňujúci vyhľadať väzbu k jednotlivým požiadavkám. Verifikáciou dokumentácie zaisťujem jej úplnosť, súlad s požiadavkami a konzistentnosť.

Pre verifikáciu jednotlivých procesov je možné využiť techniky statickej analýzy, čo je buď manuálna revízia rôzneho stupňa formálnosti alebo kontrola využívajúca nástroje pre automatickú statickú analýzu. Pri potvrdzovaní súladu produktu so špecifikáciami sa môže využiť tiež testovanie formou dynamickej verifikácie. Týmto spôsobom sme schopný nájsť odchýlky od špecifikácií.

Rozdiel medzi statickou a dynamickou analýzou je v bode ich záujmu. Statická analýza skúma produkt ako taký naproti tomu dynamická analýza skúma chovanie a vlastnosti produktu.

3.1.1 Statická analýza

Statická analýza skúma manuálne alebo pomocou rôznych nástrojov bez toho aby došlo k spusteniu produktu. Je to typický verifikačný proces a môže prebiehať už od raného štádia vývoja. Nejde len o testerov, ktorí skúmajú jednotlivé produkty a ich špecifikácie, ale ako už bolo spomenuté vyššie zaraďujeme tu tiež automatickú statickú analýzu, ktorej nástroje sú už v dnešnej dobe súčasťou integrovaných vývojových prostredí. Väčšinou je táto analýza zameraná na zdrojový kód. Manuálna revízia sa skladá z niekoľkých špecifikovaných techník. Ide o neformálnu revíziu, čo je veľmi rýchly a nízko-nákladový proces. Ide o stretnutie autora s jedným či viacerými kolegami, kde spolu prechádzajú jednotlivé časti dokumentu a hľadajú možné defekty, či problematické miesta. Nájdené nezrovnalosti sa okamžite opravujú. Ďalšou formou manuálnej revízie je štruktúrované prechádzanie dokumentu či kódu. Schôdzku na danú problematiku riadi sám autor. Za prítomnosti osôb dostatočne znalých, prechádza konkrétny dokument (kód) a spoločne nielen že odhaľujú defekty, ale zvažujú iné možné spôsoby implementácie a rozširujú znalosti o produkte v rámci tímu. Za formálnu revíziu už môžeme označiť technickú revíziu, ktorá hodnotí, či produkt spĺňa dané požiadavky pre svoje použitie a teda či odpovedá špecifikáciám. Výstupom je zoznam nájdených nezrovnalostí a tiež odporučenie, či by mal byť daný produkt prepracovaný, zamietnutý alebo akceptovaný.

Veľmi formálnym procesom je inšpekcia, inak nazývaná ja Faganovská inšpekcia (vyvinul Michael Fagan v roku 1976). Ide o plánovanú aktivitu s presne určenými pozíciami – moderátor, autor, oponenti, sprievodca, zapisovateľ. Každá z pozícií má presne určenú činnosť, preto osoby do inšpekcie zapojené musia byť v tejto oblasti znalé. Z tohto dôvodu sa tento postup používa len v naliehavých prípadoch napriek tomu, že ide o veľmi efektívnu metódu.

3.1.2 Dynamická analýza

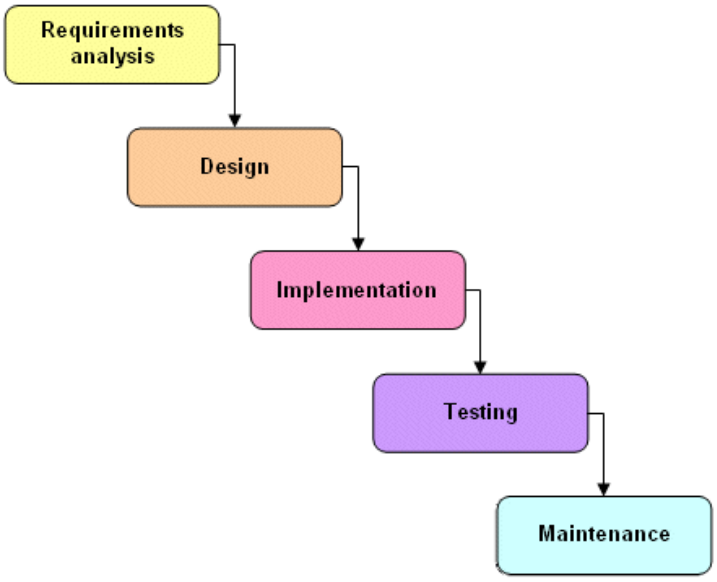
Dynamická analýza využíva k skúmaniu špeciálne k tomu určené nástroje. Umožňuje detailne analyzovať správu pamäte a výkon systému, čím zisťuje problémy, ktoré by boli inak len ťažko odhaliteľné. Dynamická analýza predstavuje testovanie v tých najrôznejších podobách.

3.2 Validácia

Oproti tomu validáciou odpovedáme na otázku „vytváram správny produkt?“ Inými slovami zisťujem , či produkt spĺňa požiadavky zákazníka a teda musí byť z pohľadu zákazníka i testovaný. Samozrejme sa môže stať, že verifikovaný produkt nebude validovaný a preto sa zákazník musí zúčastniť testovania čo najskôr. Úspešný verifikačný proces nikdy nie je pre kvalitu produktu dostačujúci, pretože len samotný zákazník je schopný správne overiť všetky požadované vlastnosti vytváraného produktu. Validácia sa však nemusí vykonávať len na kompletnom produkte, ale i na jednotlivých čiastkových produktoch.

Na zobrazenie validačných a verifikačných aktivít existujú rôzne modely. Jedným z nich je tiež takzvaný vývojový model vodopádu „waterfall model“.

Obrázok 2Waterfall model

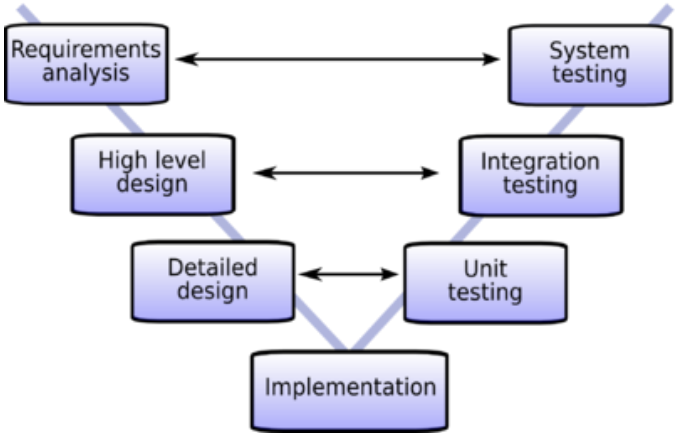


Zdroj: https://www.technologyuk.net/computing/sad/waterfall-model.shtml

Z tohto modelu potom vychádza V-model [12]. V modely vodopádu každá fáza nasleduje až po dokončení predchádzajúcej fáze. To však so sebou prináša problém ak došlo k chybnej interpretácii požiadavku, potom bude daný problém odhalený až vo fáze akceptácie zákazníkom. Pri oprave tak bude potrebné prejsť znovu celý cyklus, čo je však veľmi nákladné. A preto bol tento model upravený na V-model, ktorý výstup

z každej fázy overuje, a práve to umožňuje odhaliť chyby včas a tým sa znižujú náklady na opravu.

Obrázok 3 V-model



Zdroj: https://buildingstrongerpartners.wordpress.com/tag/v-model/

Významným rysom V-modelu je účasť aktivít testovania od začiatku projektu a súbežne s vývojom, čo sa týka predovšetkým produkcie artefaktov testovania (napr. behom fáze analýzy požiadavku ide o návrh akceptačného testu a podobne).

Hlavne pri kritických projektoch môže byť vyžadované, aby verifikáciu a validáciu systému vykonávala nezávislá tretia strana.

3.3 Meranie kvality

Keďže sú softvérové projekty spojené s obrovskými investíciami a rizikami, je potrebné sa uisťovať, že sú minimalizované riziká, vývoj softvéru smeruje k žiadaným výsledkom, výsledky prídu včas, v rámci daného rozpočtu a v očakávanej kvalite. Všetky problémy je potrebné v čas identifikovať a na problémy reagovať čo najrýchlejšie. Testovací tým sa zaoberá vedľa samotného skúmania produktu i meraním kvality, vyhodnotením výsledkov a reportovaním týchto informácií ostatným osobám. Meranie kvality je dôležité z rôznych dôvodov. Ide v prvom rade o zistenie aktuálneho stavu, sledovanie postupu oproti plánu, odhalenie projektových rizík, motivovanie pracovníkov k lepším výsledkom, či zabránenie chaosu, pretože oblasť, ktorá nie je meraná, väčšinou upadá, pretože je v skutočnosti nestrážená a neriadená.

Ako prvý krok je potrebné odpovedať na otázku, čo všetko môžeme/máme merať? Predmet nášho merania je závislý od toho, kto a aké informácie o kvalite potrebuje a k čomu by mu mali slúžiť. Jednotliví ľudia by mali najskôr sformulovať otázky, na ktoré potrebujú poznať odpovede a k tomu sú potom vybrané potrebné metriky. Vyhodnocovaním výsledkov testovania je možné odpovedať na otázky týkajúce sa hlavne:

- Kvality / nekvality produktu
- Kvality / nekvality vývoja
- Kvality / nekvality testovania

Výber metriky by mal odpovedať účelu aby bola zaistená kvalita s čo najmenším úsilím a zároveň aby bolo dostatočne minimalizované riziko. Veľmi dôležité je sledovať kde sa práve nachádzame oproti tomu, kde by sme sa mali nachádzať a to či už z hľadiska rozpočtu, času alebo vykonaných a akceptovaných úkonov. Nedostatočnú kvalitu, hrozbu prekročenia rozpočtu či hroziace oneskorenie sa zisťujeme pri kvalite vstupov v oblasti projektových rizík. Oblasti nad ktorými nemáme priamu kontrolu, ide hlavne o výkon tretích strán, je rovnako opodstatnené sledovať.

3.4 Ako meriame kvalitu

Ako druhé, po voľbe predmetu merania, prichádza na rad voľba spôsobu merania. V tomto prípade sme obmedzení dátami, ktoré máme k dispozícii alebo tým, aké údaje môžeme sledovať. Je veľmi dôležité vedieť aké informácie sú na danom projekte uchovávané, aké sa používajú nástroje, metodiky a dohodnuté pravidlá spolupráce dodávateľa zákazníka. Prevažná väčšina údajov o kvalite pochádza zo systému pre správu hlásenia o defektoch systémov pre riadenie testovania. Ďalšie informácie môžu pochádzať zo systémov pre vykazovanie práce zamestnancov daných účtovných systémov alebo z vývojových systémov, ktoré sledujú, aké množstvo zdrojového kódu bolo pri vývoji zmenené čí pustené behom testu.

Metriky môžeme rozdeliť do dvoch skupín podľa toho z ako presných dát dané meranie vychádza. Takzvané tvrdé metriky sú založené na presných, objektívnych hodnotách ako je napríklad počet nájdených defektov, počet vykonaných testov a podobne. Väčšinou sa používajú tvrdé metriky keďže sú presnejšie a nie sú ovplyvnené

osobou, ktorá ich vykonáva. Pri mäkkých metrikách ide skôr o subjektívne hodnotenie ako je napríklad spokojnosť zákazníka. Výsledky však nemusia byť presné, čo je zrejmé, pretože každý užívateľ môže byť v momente testovania v inom duševnom rozpoložení a tiež každý môže danú situáciu vnímať inak. Napriek tomu však je i subjektívny názor podstatným meradlom, keďže ako znelo na začiatku v snahe definovať kvalitu, je to niečo čo dokážeme identifikovať, no nie rozumne definovať.

Metriky členíme a kategorizujeme podľa dát z ktorých vychádzajú:

3.5 Metriky založené na hlásení defektov

Tieto metriky nám dávajú prehľad o nedostatkoch systému a ich odstraňovaní. Zisťujeme informácie ako počet objavených defektov, priemerná rýchlosť objavenia ďalšieho defektu pri testovaní, rýchlosť opráv defektov, doba za ktorú sa nám pravdepodobne podarí do stavu zlučiteľnosti defektov s uvoľnením produktu do produkčného prostredia.

Z týchto informácii vyvodzujeme druhy metrík:

3.5.1 Počet hlásení defektov podľa stavu, priority, oblasti

Poskytuje rýchly prehľad o počte defektov, ich závažnosti a informácia o tom, v akom stave sa nachádza ich riešenie. Rovnako rýchly prehľad o kvalite a momentálnej pripravenosti na nasadenie, za predpokladu, že všetky testy boli vykonané, prípadne koľko ohlásených defektov nám ostáva vyriešiť, než budeme môcť splniť kritériá pre nasadenie. Bohužiaľ neodkrýva príčiny zistenej chybovosti

3.5.2 Počet hlásení defektov v čase

Množstvá defektov v rôznych časových obdobiach sú zaznamenávané do tabuľky, z čoho sú následne vytvárané grafy. Sleduje sa napríklad:

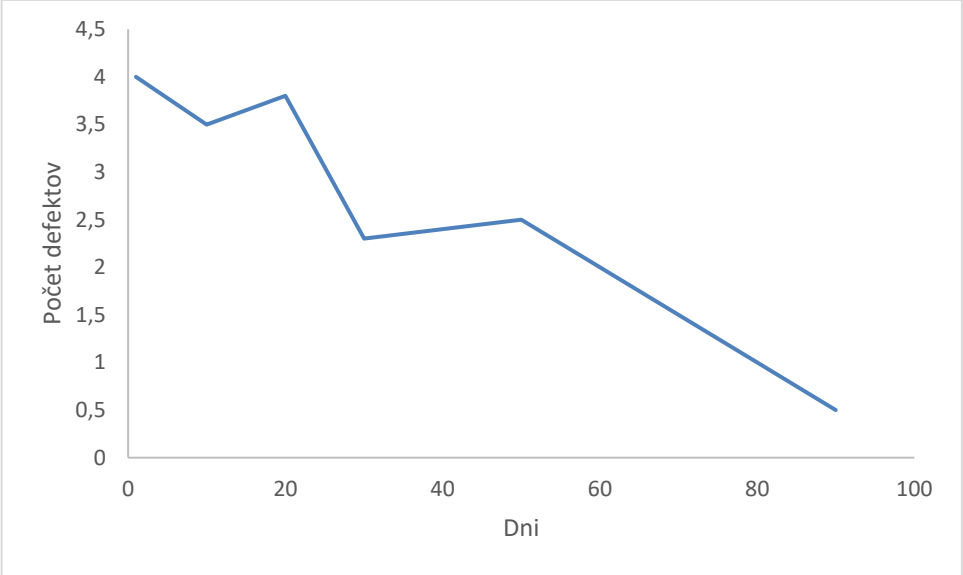
- Počet všetkých ohlásených defektov
- Pôvodne predvídaný počet objavených defektov
- Počet uzavretých defektov
- Počet neuzavretých defektov
- Počet neuzavretých defektov rozdelených podľa priority
- Počet neuzavretých defektov podľa oblastí

Táto metrika nám pomáha indikovať nedostatočnú kvalitu a predvídať z nej plynúce dôsledky: nedodržanie termínu a navýšenie rozpočtu. Nevýhodou tejto metriky je, že meranie je ovplyvnené výraznejšími zmenami v dobe, ktorá sa vyhradzuje len pre testovanie. Tento problém sa najviac prejavuje pri sledovaní trendu. V takom prípade sa dajú použiť miesto dní a týždňov skôr človekodni alebo človekotýždne.

3.5.3 *Rýchlosť nájdenia defektu*

Rýchlosť nájdenia defektu je vnímaná ako pomer počtu nájdených defektov ku počtu hodín strávených ich hľadaním. Môžeme tak zrovnávať výkon rôznych testovacích techník alebo výkon členov testovacieho tímu. V prípade dlhodobého sledovania vývoja tejto hodnoty získame informácie o zmenách kvality produktu alebo o zmenách účinnosti použitého spôsobu testovania. Pomáha nám tiež odpovedať na otázku, či za posledné obdobie bolo testovanie finančne výhodné a či je výhodné v testovaní pokračovať. K tomu aby sme mohli využiť túto metriku, však potrebujeme údaj o presnom počte človekodní strávených len na testovaní.

Obrázok 4 Vývoj rýchlosti nájdenia chýb v čase



Zdroj: Autor

3.5.4 *Intenzita chýb*

Intenzita chýb zisťuje priemerný počet chýb v jednotlivých častiach systému vzhľadom k veľkosti systému. Ukazuje na aké konkrétne časti sa v testovaní zamerať,

ktoré časti sú najproblémovejšie. Slúži tiež ako podklad pre vyhodnocovanie chybovosti. Ak nastávali problémy v minulosti je veľká pravdepodobnosť, že to bude pokračovať aj naďalej. Nevýhodou tejto metriky je, že vyžaduje zhodný stav vykonaných testov pri všetkých častiach systému, čo veľmi obmedzuje jej použitie behom testovania. V tomto prípade sa jedná o najdôležitejšiu metriku.

3.5.5 Účinnosť testovania – percento detekovaných defektov

Ide o podiel počtu defektov objavených behom testovania a celkového počtu defektov Zisťujeme teda koľko percent všetkých defektov bolo objavených testovacím tímom. Metrika je zameraná na to, do akej miery bolo predchádzajúce testovanie efektívne. Čím neskôr je toto percento merané, tým je jeho hodnota presnejšia. Vysoké percento detekovaných chýb nemusí nutne znamenať vysokú efektivitu daného testovania, ale môže tiež znamenať nízku efektivitu nasledovného testovania. Najpresnejšie meranie je teda niekoľko mesiacov po predaní do produkcie.

3.5.6 Účinnosť testovacieho úsilia

Táto metrika je veľmi podobná účinnosti testovania. Zmyslom testovania však nie je defekty len hľadať, ale tiež poskytnúť dostatočnú podporu pre ich odstránenie. Účinnosť testovania je teda rovná podielu počtu opravených defektov pred nasadením do behu celkového počtu defektov nájdených v systéme celkovo.

3.5.7 Priemerná cena nájdenia a odstránenia defektu

Priemerná cena je podiel nákladov na testovanie a počtu nájdených defektov. Priemerná cena odstránenia defektu je rovná podielu nákladov na odstránenie defektu a počtu odstránených defektov. Do nákladov na testovanie započítavame tiež náklady na riadenie testovania, prípravu testovacích prípadov, scenárov a hlásení defektov, pretože to všetko je súčasťou procesu testovania.

Tu však treba podotknúť, že sa nedá úplne presne veriť priemernej cene odstránenia defektu, keďže napríklad cena za odstránenie piatich D-defektov nemusí byť ani jedna tretina z ceny odstránenia jedného A-defektu.

3.6 Metriky založené na testoch

Patrí sem skupina metrík, ktoré využívajú hlavne výsledky vykonaných testov. Je to veľmi významná skupina z pohľadu sledovania postupu a výsledkov testovania. Potrebné dáta sú väčšinou automaticky extrahované zo systému pre riadenie testovania a sú pre manažment a okolie zrozumiteľnejšie než informácie o počte, stavoch a prioritách ohlásených defektov.

3.7 Matica pokrytia oblasti testy

Ukazuje nám, či boli testy jednotlivých častí systému spustené a s akým výsledkom. Hodnota zväčša býva udávaná v percentách. Hlavným účelom je rýchly prehľad o stave a priebehu testovania v jednotlivých oblastiach. Ide o veľmi jednoduché a prehľadné zobrazenie stavu testovania za pomoci len troch údajov. Avšak na druhej strane tu chýba zohľadnenie počtu a priorít defektov a preto, môže byť pri oblastiach s malým počtom testov zobrazenie percent mierne zavádzajúce.

3.8 Výsledky testov v čase

Tu sa zameriavame v prvom rade na vývoj množstva vy konaných, prípadne úspešných alebo neúspešných testov v čase. Podľa rýchlosti, s ktorou sa počet testov v daných kategóriách mení, je možné odhadovať, kedy bude testovanie dokončené. To je možné najlepšie sledovať na grafe, zvlášť pokiaľ skutočný vývoj porovnávame s plánovaným. Postup pri tvorbe grafu:

1. Odhadneme a naplánujeme, koľko testov má byť urobených v priebehu iterácií.
2. Stanovíme, u koľkých testov je očakávané neúspešné vykonanie, a teda nutnosť opakovania.

3. Následne postupne – obvykle po dňoch alebo týždňoch – do tabuľky zaznamenávame, koľko testov už bolo vykonaných a koľko ich bolo úspešných.

4. Vývoj sledujeme v grafe a aproximujeme, nakoľko je splnenie plánu skutočne dosiahnuteľné. Pre aproximáciu sa väčšinou používa trend vo forme tretieho polynómu.

Je vhodné sledovať rôzne kategórie testov zvlášť, keďže doba a zložitosť vykonania týchto testov sú značne odlišné. Indikujú sa tu problémy s kvalitou a dostupnosťou testovacieho prostredia. Je možné predvídať, ako dlho bude pri danom trende testovanie trvať. Nezistíme však príčinu pomalého postupu testovania. V prípade veľkých výkyvov v počte človekodní strávených testovaním je vhodné sledovať čas v človekodňoch miesto v normálnych dňoch. V opačnom prípade nie je metrika príliš presná. Treba dodať, že u akýchkoľvek odhadov budúcnosti ide vždy len o odhad, podobný napríklad dlhodobej predpovedi počasia. Samozrejme je vyžadovaná určitá skúsenosť pri analýze jednotlivých hodnôt, odhadnutí trendu či interpretácii výsledku.

Toto bol stručný prehľad základný metrík merania kvality. Existujú však aj ďalšie oblasti, ktoré je užitočné sledovať a overovať. Jednou z takýchto oblastí je napríklad sledovanie spotrebovaných človekodní oproti plánu. Človekodni sú sledované pravidelne za jednotlivé položky plánu aj za všetky aktivity dohromady. Dáta sú zaznamenávané do tabuľky. Je nutné uvádzať koľko bolo v danom mesiaci vynaložených človekodní na riadenie projektu, na aktivity mimo testovania, na prípravu testov, čisto na testovanie a reportovanie defektov a na školenia.

Týmto spôsobom je možné identifikovať vysoko nákladné činnosti. Jednoducho tak získame podklady pre hľadanie úspor na projektoch, databázu pre tvorbu budúcich dosiahnuteľných plánov a získame možnosti včasného rozpoznania rozpočtových problémov. Z týchto dát je možné analýzou zistiť vplyv nových postupov na rozpočet projektu.

Ďalšou oblasťou je dostupnosť testovacieho prostredia. Pokiaľ sa na projekte vyskytujú časté výpadky testovacieho prostredia ohrozujúce plán, môže byť užitočné sledovať dostupnosť testovacieho prostredia v jednotlivých dňoch, prípadne tiež príčiny nedostupnosti. Ide o zdôvodnenie a ospravedlnenie zmien plánu.

Subjektívnym hodnotením kvality systému užívateľmi alebo testermi sú takzvané „dotazníky na kvalitu“. Tie vypovedajú o dosiahnutej kvalite produktu a procesu

testovania. Porovnávajú sa tu názory na kvalitu produktu medzi testermi pri akceptácii a užívateľmi. Dotazníky zisťujú spokojnosť užívateľov a zisťujú nedostatky v procesoch vývoja. Táto metrika je však príliš subjektívna a hodnotenie je väčšinou spätné, tj. Produkt je už v tej dobe používaný.

4. Nákladová analýza

Táto kapitola sa venuje aspektom, ktoré ovplyvňujú náklady na kvalitu softvéru. Pokiaľ je testovanie softvéru dostatočné a sú použité primerané metriky, procedúry a nástroje na podporu testovania, náklady by mali poklesnúť a kvalita softvéru by sa mala zvýšiť. To však nie je vždy pravda. Výrazne redukovať náklady môže včasné odhalenie chyby.

[1][6][14][17]

4.1 Náklady na odstránenie chyby v závislosti na etape životného cyklu

Tabuľka 1 Náklady na odstránenie chyby v závislosti na etape životného cyklu. Baziuk 1995 vs. Boem 1976.

Etapa životného cyklu	**Náklady na odstránenie chyby** *[X je jednotka nákladov. Etapa životného cyklu vyjadrená v človekohodinách alebo peniazoch.]*	
	Štúdia p. Baziuka - 1995	**Štúdia p. Boehma - 1976**
Požiadavky	1x	0,2x
Dizajn		0,5x
Kódovanie		1,2x
Testovanie modulov		
Systémové testovanie	90x	5x
Inštalačné testovanie		
Akceptačné testovanie	440x	
Prevádzkovanie a údržba	470x-2900x	

Zdroj: https://www.mtf.stuba.sk/docs//internetovy_casopis/2005/5/tanuska.pdf

Ako je možné vidieť v tabuľke 1, je dôležité odhaliť chybu v čo najkratšom čase, pretože čím neskôr ju odhalíme, tým budú náklady na jej odstránenie vyššie. Najnákladnejšie je objavenie chýb vo finálnej fáze vývoja.

Znamená to teda, že náklady na opravu chýb logaritmicky rastú s dobou odhalenia chyby. Softvér obvykle ponúka širokú škálu funkcionalít a preto sa môže stať, že nie je možné otestovať všetky možné kombinácie vstupov a výstupov. Vzhľadom k tomu, je nutné použiť metódy testovania, ktoré pokryjú čo najširšiu funkcionalitu s minimálnym počtom testovacích prípadov. K tomuto účelu slúžia rôzne techniky ako pre manuálne, tak aj pre automatické testovanie.

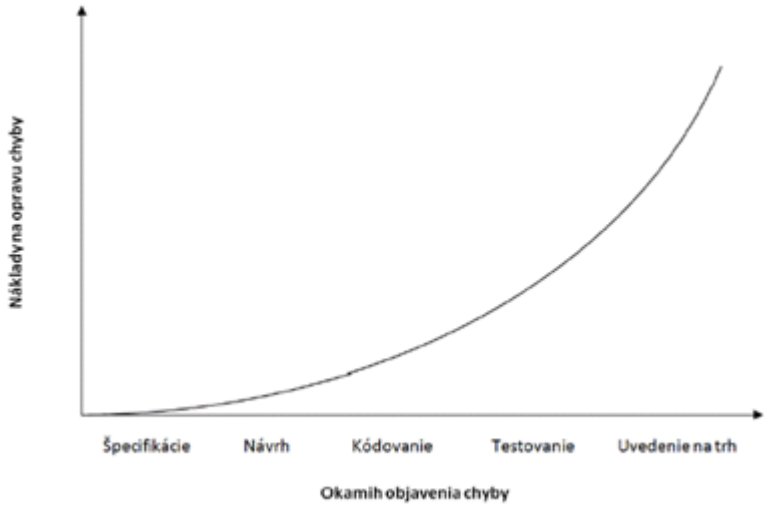
Automatizácia testovania zefektívňuje a zrýchľuje celý proces, hlavne v prípade často sa opakujúcich rovnakých testovacích scenárov. Automatické testovanie je však nutné starostlivo naplánovať, pretože i keď zefektívňuje proces testovania, taktiež má svoje úskalia. Je preto na zváženie do akej miery je možné automatizovať tak, aby to stále bolo efektívne a výstupy tohto testovania boli použiteľné. Vhodné sú hlavne pri jednotkovom testovaní, kde dokážu veľmi rýchlo otestovať algoritmus aj pri veľkom množstve vstupných dát. Avšak v niektorých oblastiach je manuálne testovanie skutočne nenahraditeľné, čím je napríklad testovanie užívateľského rozhrania. Rovnako sú s automatickým testovaním spojené vyššie vstupné náklady ako je to pri manuálnom testovaní, avšak v dlhodobých a stabilných projektoch, kde sú opakujúce sa procesy dochádza z dlhodobého hľadiska ku zníženiu nárokov na finančné prostriedky.

Povinnosťou testerov je otestovať softvér kompletne, to znamená, že nie len funkcionality, ktoré by mali fungovať, ale musia sa zamerať tiež na objavovanie skrytých chýb v aplikácii. Pokiaľ sa počas doby vývoja projektu nejaká chyba neodhalí, môže to znamenať v závislosti na type chyby skutočné predraženie projektu. Tester preto musí disponovať analytickým myslením, ktoré mu pomôže chápať aplikáciu komplexne a vytvárať scenáre, ktoré dopomôžu k objaveniu chýb aplikácie. Tester tiež skúma aplikáciu z pohľadu zákazníka a snaží sa dosiahnuť maximálnu možnú kvalitu softvéru tak, aby splnil predstavy zákazníka.

Chyby je možné objaviť na začiatku alebo v inej fáze vývoja. Samozrejme ako už bolo spomenuté, čím neskôr je chyba objavená, tým vyššie sú náklady, pretože množstvo prostriedkov, ktoré je potrebné vynaložiť na opravu chyby sa zvyšuje úmerne s časom. Náklady spojené s opravou chyby na začiatku vývoja sú výrazne nižšie ako tie, ktoré sú objavené samotným zákazníkom až po zavedení do plnej prevádzky. Navyše chyby objavené zákazníkom v priebehu používania dodaného softvéru môžu viesť k nespokojnosti zákazníka a následnému poškodeniu dobrého mena spoločnosti, prípadne k strate stáleho zákazníka. Je nutné sa takýchto chýb vyvarovať i keď to nemusí

byť jednoduché, hlavne v prípade nepochopenia požiadaviek zákazníka, kedy softvér absolútne nemusí spĺňať jeho predstavy.

Obrázok 5 Náklady na opravu chýb softvéru



Zdroj: http://cstudio.sk/skola/dp/

Testovanie, ako proces, ktorý má tieto chyby odhaľovať, sa uskutočňuje na rôznych úrovniach s cieľom postupne zachytiť všetky vzniknuté chyby. Rôzne úrovne testov sa tak snažia odhaliť problémy spojené s rôznymi fázami vývoja softvéru. Avšak určité percento chýb je možné odhaliť aj pred samotným uvedením softvéru do prevádzky, napríklad využitím formálnych revízií, pri ktorých sa skúma programový kód z pohľadu zachovania štandardov, konvencií, zásad programovania a výskytu rôznych iných nedostatkov.

Ako už bolo spomenuté, priemerná cena nájdenia defektu je rovná podielu nákladov na testovanie a počtu nájdených defektov. Priemerná cena odstránenia defektu je rovná podielu nákladov na odstránenie a počtu odstraňovaných defektov.

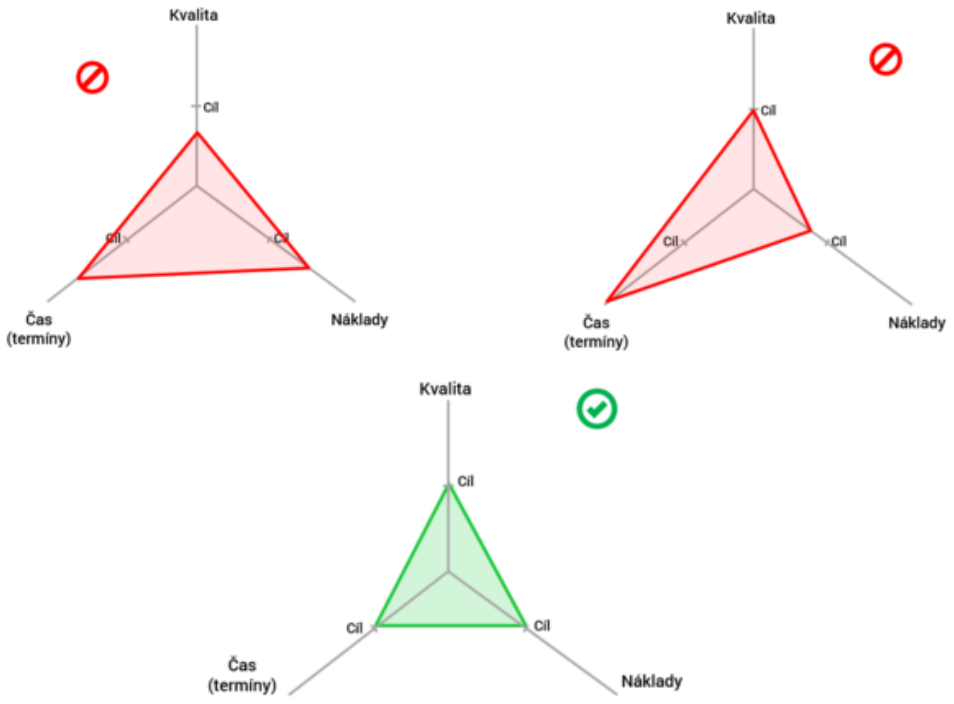
Do nákladov na testovanie tu môžeme počítať aj náklady na riadenie testovania, prípravu testovacích prípadov, scenárov a hlásenia defektov, pretože to všetko je súčasťou procesu testovania. Pri výbere toho, čo všetko bude do nákladov zahrnuté, záleží na tom k akému účelu chceme metriku využívať.

Ak spočítame celkové náklady na defekt pri jeho objavení a odstránení behom fáze testovania a porovnáme výsledok oproti celkovým nákladom na defekt pri jeho objavení v celej produkcii, môžeme rozdiel týchto dvoch hodnôt použiť k výpočtu, akú veľkú úsporu testovanie prináša. To môže byť zaujímavé hlavne v prípade, keď bolo z dôvodu nedokončenia testov odložené nasadenie, pretože je možné následne určiť, aká veľká čiastka bola ušetrená.

4.2 Trojuholník kvality

V plánovaní testovania existuje tzv. trojuholník kvality. Základnými atribútmi sú doba dodania, cena a kvalita výsledného produktu. Ideálna situácia je – okamžité dodanie, nulová cena a vysoká kvalita. Táto situácia je však nedosiahnuteľná, preto je potrebné nájsť vhodný kompromis. Je zrejmé, že vždy môžeme vyvážiť len dva atribúty na úkor tretieho. Ak je napríklad nutné znížiť náklady, musíme odpovedajúcim spôsobom navýšiť dobu dodania alebo znížiť kvalitu, vždy však musí byť vykonaná zmena vyvážená.

Obrázok 6 Trojuholník kvality



Zdroj: https://managementmania.com/sk/magicky-trojuholnik-projektoveho-riadenia

„Trojuholník projektového riadenia v praxi: Riadenie projektov prináša v praxi zvyčajne rôzne komplikácie, a to sa týka aj tých najlepšie naplánovaných. V praxi potom dochádza k porušeniu jedného zo spomínaných parametrov. Najčastejšie dochádza ku oneskoreniu harmonogramu (čas), k prekročenie nákladov (rozpočet projektu), niekedy sa pri snahe dodržať tieto dva zhorší kvalita výstupov. Každá z týchto situácií je pre zákazníka zla - neskoro dodaný výstup projektu, hoci je kvalitný a za pôvodnú cenu môže spôsobiť rovnaké problémy ako nekvalitné výstup napriek tomu že je dodaný včas.“ [15]

5. Rozbor a vyhodnotenie merania kvality softvéru

Táto kapitola predstavuje praktickú časť bakalárskej práce. Je venovaná rozboru metodík testovania, ktoré sa pre testovanie používajú v reálnych IT projektoch. Zaistením dostatočného testovania softvéru, a teda čo najefektívnejšie využitie jednotlivých metodík zároveň dáva priestor na meranie samotnej kvality. Kapitola je rozdelená do dvoch hlavných častí. Prvá časť sa venuje rozboru merania kvality v prostredí IT projektu, ktorého som osobne bola určitú dobu súčasťou. Ide teda o rozbor z pohľadu skutočného užívateľa jednotlivých metodík. V tejto kapitole sa snažím vyzdvihnúť dôležitosť niektorých metodík a poukázať na výhody, ktoré ich používaním získavame. Zároveň porovnávam ako by bol projekt ohrozený a koľko by jeho chod stál v prípade, že by jeho kvalita testovaná nebola a naopak, koľko stojí overovanie kvality softvéru a ako znižuje rôzne hrozby vyskytujúce sa pri implementácii softvéru.

Ako už bolo v práci spomenuté, pojem kvalita je ťažké jednoznačne definovať. V skratke by sa dalo povedať, že kvalitný produkt je produkt, ktorý uspokojí úplne každého. Problém je však v tom, že každý má na kvalitu iné požiadavky. A preto ak by sme chceli takýto produkt vytvoriť boli by sme nútený analyzovať požiadavky všetkých stakeholderov bez ohľadu na to, koho názor je dôležitejší. Analyzovať by to možné bolo, problém však nastane až po spracovaní týchto požiadaviek. Dôkladne zanalyzované požiadavky by síce nemal byť problém implementovať, avšak častokrát sa stáva, že očakávaný výsledok bol iný ako ten, ktorý sa v skutočnosti dosiahol. Pre niekoho môže byt tento výsledok dostačujúci pre iného môže byť úplnou katastrofou.

A tu sa dostávame do bodu zlomu, kedy sme nútení prikloniť sa buď na jednu alebo na druhú stranu. Tým pádom strácame možnosť vyrobiť dokonalý produkt. Problém však môže nastať už oveľa skôr. V takmer každom projekte dôjde k nepochopeniu požiadaviek od zákazníka. Vina nemusí byť ihneď pripisovaná analytikovi, častokrát sám zákazník nie je schopný popísať svoje požiadavky dôkladne alebo na niečo zabudne alebo si určité kroky sám neuvedomuje.

Nejde tu však len o analýzu, chybovosť sa vyskytuje v každej fázy projektu. Keďže už vieme, že neexistuje cesta k dokonale kvalitnému produktu, snažíme sa k tejto kvalite aspoň čo najviac priblížiť. To by nám malo zaručiť správny výber metrík, rozumné rozdelenie tímových rolí, dobrá koordinácia a spolupráca v tíme, pravidelné

vyhodnocovanie stavu, dobrá komunikácia a podobne. Ak by toto všetko fungovalo na špičkovej úrovni, o kvalite výstupov by sme nemuseli pochybovať.

A práve preto sa meranie kvality v jednotlivých projektoch líši. Každý IT projekt má iné cieľové zameranie a tiež rôzne podmienky pre realizáciu projektu čo značne ovplyvňuje to, akým spôsobom sa kvalita produktu bude zaisťovať a tým pádom aj merať.

5.1 Projekt č.1

Projekt číslo jedna je zasadený do bankového prostredia. Projektový tím má menší počet členov. Je to hlavne z toho dôvodu, že softvér je dodávaný externou vývojárskou firmou. Role v projektovom tíme sú: Projektový manažér, produktový manažér, scrum-master. architekt, aplikačná podpora, aplikačná integrácia, test manažér a tester. Projekt aplikuje agilnú metodiku, členovia sa stretávajú na pravidelných stand-up meetingoch, ktoré sa konajú každý deň, celkový stav projektu a zhodnotenie progresu je vykonávané na status-meetingoch, ktoré slúžia ako odrazové body pre plán na ďalšie obdobie a osobné problémy jednotlivých členov, či už sociálneho alebo technologického charakteru, sú komunikované na pravidelnej retrospektíve. Tím dodržiava pravidlá agilnej metodiky scrum[3] a jednotlivý členovia si plnia svoje povinnosti. Časový harmonogram nie je narušený. Tým je schopný rýchlo reagovať na náhle zmeny.

Projekt je naplánovaný na jednotlivé release[4]. Hneď na začiatku je daný jasný release plán, čo môžeme považovať za víziu celého projektu, a ten obsahuje počiatočné požiadavky na dané vlastnosti systému. Počas celého cyklu behu projektu sa požadované vlastnosti modifikujú a práve preto je nutnosťou schopnosť rýchlo na zmenu reagovať a to či už na strane vývojárov alebo analytikov, testerov a dizajnérov. Projekt je postavený tak, aby tieto situácie zvládol, a aby zmena alebo pridávanie nových požiadavkou počas vývoja bola vyhovujúca. Keďže hlavnou myšlienkou celého projektu je vytvoriť skutočne užívateľsky na pohľad prívetivé, prehľadné a nenáročné prostredie, je potrebné počas vývoja skúmať pohľady užívateľov na rôzne verzie softvéru a zvoliť tú najvýhodnejšiu variantu.

[3] Scrum: iteratívna a inkrementálna metodológia agilného vývoja softvéru používaná na riadenie produktového vývoja
[4] Release: obdobie jednej iterácie po nasadenie verzie softvéru

Softvér sa implementuje v jednotlivých sprintoch. Pred každým sprintom sa do Sprint Backlogu zaradia požiadavky na zmeny. A preto je nutné rovnaké metódy testovania vykonávať opakovane – koľko verzií sa nasadí, toľkokrát celý proces testovania prebehne.

Prvé dve metódy overovania kvality, tj. Testovanie v priebehu vývoja a Jednotkové testy, sa priamo v projekte nepoužívajú. Je to z toho dôvodu, že softvér je vyvíjaný inou, nezávislou organizáciou. Najviac sa apeluje na UAT testy, ktorých výstupy by mali byť čo možno najkvalitnejšie.

V popisovanom projekte nie sú jednotlivé metriky na overovanie kvality použité ako také, ale sú zakomponované v rámci metód testovania. V každej metóde sa predpokladá, že zodpovedná osoba dbá na overenie kvality nezávisle na tom, akým to vykoná spôsobom. Najdôležitejším meradlom sa však v tomto projekte stáva výsledok nezávislého testovania, ktoré ukazuje celkovú použiteľnosť softvéru. Počet hlásení defektov či už v čase alebo podľa priority kontroluje testovací manažér, ktorý na základe informácii z jednotlivých metodík vytvára reporty a tie raz mesačne prezentuje pred hlavným vedením tímu.

5.1.1 Testovanie v priebehu vývoja

Predpokladá sa, že tento typ testovania je vykonávaný externou organizáciou, ktorá softvér dodáva. Členovia projektového tímu nezisťujú, či toto testovanie prebehlo, predpokladá sa však dodanie softvéru v dostatočnej kvalite a vyvinutého podľa daných požiadaviek.

5.1.2 Jednotkové testy

Rovnako ako v predchádzajúcej metóde aj toto testovanie sa očakáva pri vývoji dodávaného softvéru a preto sa v rámci daného projektu nevykonáva.

5.1.3 Integračné testy

Priebeh integračného testovania má na starosti test manažér. Pre tento druh testovania sú na projekte určení dvaja špecialisti, ktorý sa venujú len samotnej integrácii. Integračné testy prebiehajú pravidelne počas celého vývoja softvéru, vždy po nasadení novej verzie. Je veľmi dôležité vykonať integračné testy ako prvé, aby sa predišlo veľkej chybovosti softvéru počas akceptačného testovania. Môže sa totiž stať, že pokiaľ neprebehnú

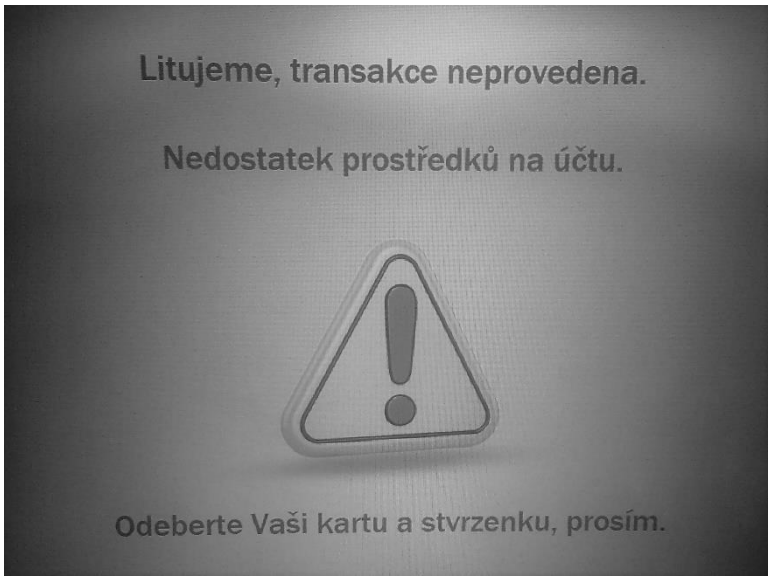
integračné testy, softvér sa bude chovať inak ako sa očakáva a tester to v akceptačných testoch vyhodnotí ako chybu dizajnu prípadne funkcionality, napriek tomu, že môže ísť o chybnú komunikáciu medzi jednotlivými komponentami vo vnútri aplikácie.

5.1.4 UAT testy a systémové testy

Rozdiel medzi systémovými a akceptačnými testami nie je v tomto projekte príliš veľký. Tento druh testov vykonáva tá istá skupina testerov. Testuje sa na základe jednotlivých požiadaviek managementu a tak po každom nasadení prebehne postupne ako systémové tak aj akceptačné testovanie. Táto skupina testerov sa nazýva UAT (user acceptance testing) a teda celá fáza testovania je nazývaná UAT testy.

I keď za dôležité musíme považovať všetky fázy, testovanie touto formou je najdôležitejšou fázou počas životnosti projektu. Tu prebehnú všetky regresné testy, otestuje sa dizajn a zhodnotí sa celkové pôsobenie a funkčnosť softvéru. Ak by doposiaľ neboli objavené chyby softvéru, tu už odhalené byť musia.

Obrázok 7 Chyba: zobrazenie nedostatku prostriedkov na účte napriek dostatočným prostriedkom – regresný test



Zdroj: Autor

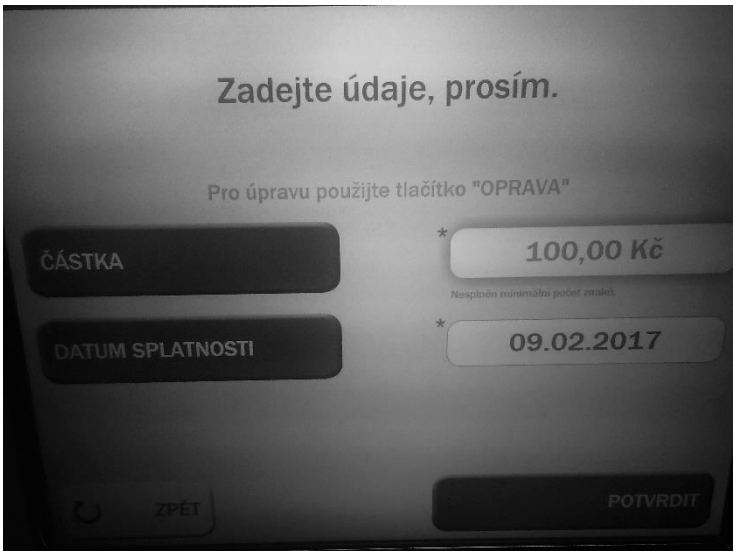
Čo sa regresných testov týka, tie majú za úlohu overiť funkcionalitu softvéru. Na začiatku projektu sa pripravia regresné testovacie scenáre, ktoré sa počas projektu využívajú aj v ďalších iteráciách a podľa potreby sa upravujú. Testovacie scenáre sa vytvárajú na základe business požiadaviek, ktoré sú špecifikované v Enterprise Architekt (ďalej EA). EA zachytáva každú regresiu a celkový tok udalostí, ktoré môžu v aplikácii

nastať. Sú tu teda dostupné diagramy, ale tiež šablóna obrazoviek, ktoré reprezentujú ako by mala aplikácia vyzerať v užívateľskom prostredí.

Textový obsah aplikácie je definovaný v samostatnom súbore. Keďže je softvér dostupný vo viacerých jazykoch, je potrebné definované texty zaslať agentúre, ktorá sa postará o kvalitný preklad. Na základe týchto materiálov sa vytvárajú testovacie scenáre. Tester následne postupuje podľa týchto scenárov.

Ak tester nájde nezrovnalosti vystaví defekt. Pre tento prípad slúži aplikácia ALM. Na vystavenie defektu je potrebné do aplikácie zadať cestu, akou sa tester k defektu dostal, verziu na ktorej testoval, čo sa od aplikácie očakávalo a čo skutočne nastalo. Nakoniec je potrebné vyplniť severitu – tj. či je defekt veľmi vážny (fatal), vážny, ale nie je nutné ho opraviť ihneď (critical, major) alebo ide len o „kozmetickú vadu“ (minor). Vhodné je tiež k defektom pridávať prílohy, či už ide o obrazový alebo zvukový materiál alebo logy, ktoré vývojárom ukážu, čo sa dialo na pozadí aplikácie. Na to, aby sa tester k týmto logom dostal, sa musí vzdialene prihlásiť na stanicu vývojárskej organizácie, kde si jednoducho stiahne všetky potrebné informácie a priloží ich k vytvorenému defektu.

Obrázok 8 Chyba: Zvýraznenie poľa napriek zadaniu správnej čiastky



Zdroj: Autor

Po odoslaní defekt putuje k testovaciemu manažérovi, ktorý ho skontroluje, prípadne upraví a odošle vývojárom. Opravené defekty sú spísané v takzvanom Hotfix Release, ktoré po oprave posiela externá vývojárska firma späť na projektový tým. Akonáhle Hotfix testeri obdržia, sú ihneď zahájené retesty. Pokiaľ defekty nie sú opravené, alebo sú opravené nedostatočne, test manažér problém komunikuje s príslušnou osobou

dodávateľa softvéru a snažia sa spolu prísť na to, prečo problém nie je možné opraviť podľa zadaných požiadavkou a ako danú situáciu riešiť. Niekedy sa príde na to, že chyba nebola v ALMe popísaná dostatočne, prípadne chýbali prílohy, ktoré by chybu dokazovali. Prílohy tiež slúžia k lepšiemu pochopeniu defektu a preto v prípade ich vynechania, môže dôjsť k nedorozumeniam.

Projekt aplikuje metodiku scrum čo znamená, že všetky druhy testov a teda aj UAT testy prebiehajú opakovane. Tým sa zamedzí príliš vysokej chybovosti. Keďže ide o manuálne testovanie je pravdepodobné, že pri overovaní kvality môžu nastať chyby. Toto testovanie býva v určitých fázach monotónne, čo znižuje sústredenie testerov a môže sa stať, že určité chyby prehliadnu. Keďže testovanie prebieha opakovane, prehliadnuté chyby môžu byť objavené v neskorších fázach. S blížiacim sa termínom finálneho nasadenia produktu sa však nároky na kvalitu výstupov zvyšujú a preto je najdôležitejšie dostatočne overovať kvalitu práve v záverečných verziách softvéru.

Keďže tento softvér pracuje s hotovosťou je nutné overiť, či je hotovosť spracovávaná korektne. K tomuto účelu slúži virtuálna pokladňa. Tester si založí pokladňu s dostatočnou hotovosťou a vytvorí dotáciu ku konkrétnemu produktu (stroju). Následne k stroju pristúpi a nadotuje ho manuálne. Vo fáze UAT testov sleduje to, či mu produkt umožňuje bezproblémovú dotáciu a či sú jeho reakcie na dotáciu správne, prípadne či obsahuje všetky funkcionality, ktoré sú očakávané. To, čo sa deje na pozadí a či je stroj skutočne nadotovaný, sa kontroluje vo fáze integračných testov.

5.1.4.1 Testovací nástroj ALM

ALM je základným nástrojom používaný pri overovaní kvality v tomto projekte. V dnešnej dobe existuje mnoho nástrojov, procesov, spôsobov či metodík pre testovanie softvéru. To, pre aký nástroj sa rozhodneme a akou cestou sa bude testovanie uberať záleží na rôznych okolnostiach. Testovanie pomocou tohoto nástroja je vcelku užitočné a efektívne.

Application lifecycle management alebo inak ALM je aplikácia, ktorá slúži ako nástroj k zefektívneniu overovania kvality softvérových aplikácií a zahŕňa celý životný cyklus ich vzniku. Tento produkt vytvorila spoločnosť Hewlett-Packard. ALM poskytuje podporu všetkých fáz testovacieho procesu. Ponúka širokú škálu funkcionalít a preto je vhodný pre každého člena tímu. Nevyužíva sa len k zapisovaniu nájdených defektov a vytváraniu testovacích scenárov, ale naopak poskytuje skutočne širokú škálu

funkcionalít. Jeho súčasťou sú aj špeciálne nástroje pre záťažové alebo automatizované testy.

Obrázok 9 ALM

Zdroj: https://cz.pinterest.com/pin/535858055642470880/

Testeri môžu v ALMe využívať tiež nasledujúce funkcie:

- **Dashboard**

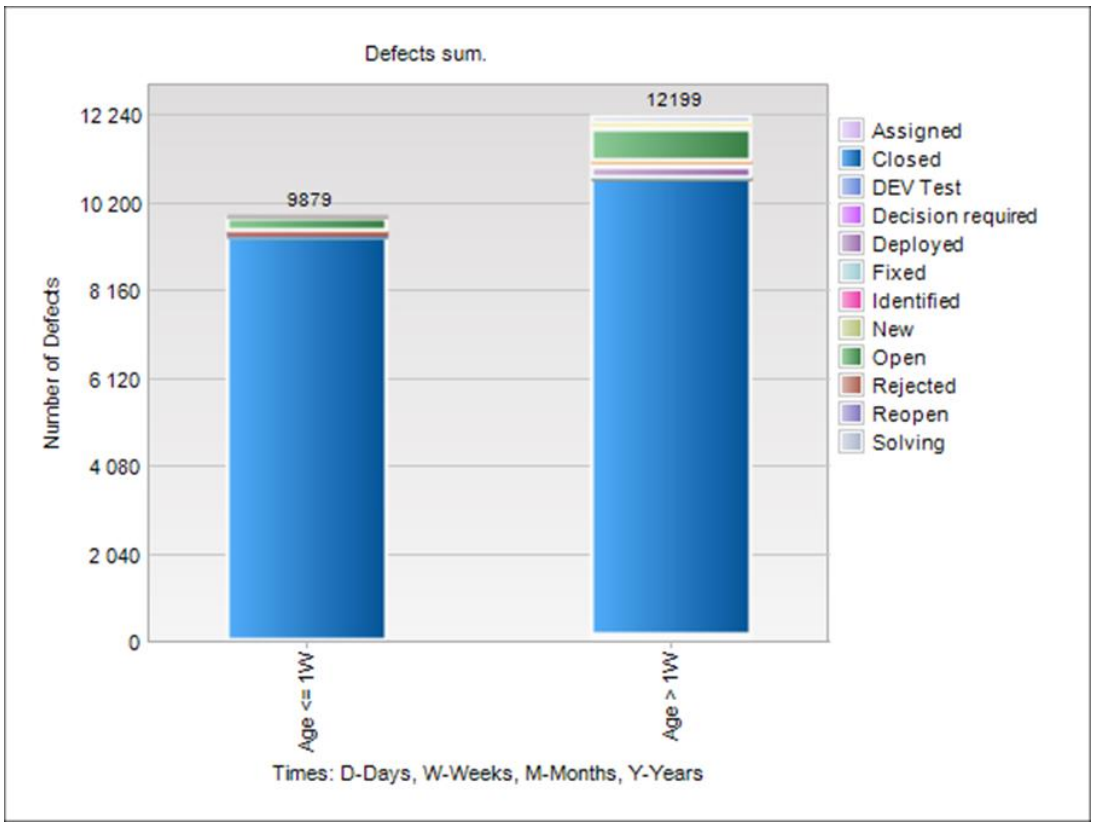
Dashboard umožňuje vytvárať a zobrazovať grafy a reporty. Testeri tu vytvárajú reporty. Tie je možné si vytvárať súkromne v súbore private a taktiež verejne v súbore public. Okrem reportov tu sú občas vytvorené aj grafy, ktoré poskytujú lepšiu prehľadnosť. Ak chce mať tester skutočne dobrý prehľad môže si v tejto sekcii vytvoriť nástenku, ktorá ponúka kompletný prehľad grafov.

- **Requirements**

„Manažment požiadaviek“ poskytuje možnosť vkladať požiadavky, ktoré môžu byť po odsúhlasení kontrolované respektíve konzultované s relevantným stakeholderom. Každej požiadavke je ihneď po vytvorení priradené ID a počiatočný stav. Stavy požiadaviek sa menia v závislosti na tom, či sa na nich pracuje. Požiadavky je možné analyzovať, prioritizovať, pridávať k nim rôzne prílohy, sledovať ich históriu a podobne. Tester si tu môže v prípade nejasností overiť niektoré skutočnosti. To však len v

nevyhnutných prípadoch. Predpokladá sa, že je všetko dôkladne zadefinované v testovacích scenároch a v dokumentoch s tým spojených.

Obrázok 10 Analysis view - dashboard



Zdroj: Autor

5.1.5 Nezávislé testovanie

Nezávislé testovanie je testovanie nestrannou osobou, ktorá dopredu presne nevie ako by mal softvér fungovať. Pre tento projekt bolo toto testovanie nevyhnutnosťou pretože dôležitou časťou kvality softvéru je práve jeho použiteľnosť. Keďže je tento druh softvéru využívaný priamo ľuďmi rôznych vekových kategórií či rôznych profesií, jedinou cestou ako vyvinúť softvér, ktorí by zvládol používať každý a ktorý by bol pre každého prívetivý a zrozumiteľný, je práve testovanie skupinou nezávislých osôb. V tomto projekte bol softvér testovaný za pomoci respondentov dodávaných agentúrou na základe vopred stanovených požiadaviek. Dôležité bolo softvér otestovať čo najrôznejšími typmi ľudí. Základnými požiadavkami teda bolo aby boli respondenti vyberaný z rôznych vekových kategórií oboch pohlaví, čo najrozličnejších druhov povolaní a nakoniec boli rozdelený do dvoch kategórií a to klientov banky a ne-klientov banky. Tento druh testovania sa

vykonával na testovacom zariadení. Toto zariadenie síce nebolo totožné so zariadeniami používanými v praxi, avšak poskytovalo všetky dôležité funkcionality, ktoré bolo potrebné otestovať. V praxi to vyzeralo tak, že každý respondent dostal provizórnu platobnú kartu a PIN kód, čo ďalej mohol využiť k požadovaným transakciám. Softvér na tomto zariadení bol ovládaný vzdialene pomocou počítača tak, aby mal respondent pocit, že sa chová úplne rovnako ako bežne používané zariadenia. Bol však obmedzený len na obrazovky, ktoré bolo nutné otestovať.

Nezávislé testovanie neprebieha po každom nasadení novej verzie softvéru. Počas pol roka môjho pôsobenia na projekte prebehlo toto testovanie dvakrát v dvoch dňoch na dvanástich respondentoch. Testovania sa zúčastňuje každý, kto sa akýmkoľvek spôsobom podieľa na celkovom testovaní softvéru. Moderátor a asistent sú prítomný počas celého priebehu nezávislých testov, pracovníci na ostatných pozíciách strávia na tomto testovaní vždy maximálne osme hodín nezávisle na tom, ktorý deň sa testovania zúčastnia.

5.1.5.1 Priebeh nezávislého testovania

Ako prvé vstúpi respondent do miestnosti so zariadením, moderátorom a prípadne asistentom moderátora, a oboznámi sa so všetkými pravidlami čo sa týka aj zvukového a obrazového nahrávania celého priebehu testovania. Moderátor položí respondentovi základné otázky týkajúce sa jeho veku, povolania a skúseností s bankomatmi rôznych bánk. Akonáhle má moderátor predstavu o tom ako respondent v bežnom živote funguje ako sa rozhoduje a aké má názory, pristupuje k ďalšej fázy a tou je priamo testovanie na pripravenom zariadení. Tu sa sleduje všetko od spôsobu vkladania karty a zadávaniu PINu až po myšlienkové pochody respondenta a jeho reakcie na rôzne obrazovky či situácie, ktoré môžu v priebehu testovania nastať. Najskôr moderátor zadá respondentovi úlohu a pozoruje ho akým štýlom ju splní. Potom vykonávajú tú istú úlohu od začiatku s tým, že moderátor sa pri každej obrazovke zastaví a vyžaduje aby respondent opísal čo vidí a vyjadril sa, čo si o jednotlivých funkciách, ktoré obrazovka ponúka, myslí. Je dôležité aby respondent v každom kroku premýšľal nahlas. Nejde tu o správne či nesprávne odpovede, ale o to, aby sa zistil myšlienkový pochod rôznych ľudí a tým sa softvér čo najviac prispôsobil ich zmýšľaniu.

Po skončení testovania na zariadení prichádza na rad tretia fáza. Respondent dostane kartičky s rôznymi prídavnými menami a päť minút čas na premýšľanie. V tejto chvíli by si mal rozmyslieť aký ma z celého softvéru pocit a priradiť mu päť vlastností, ktoré vyberie z kartičiek. Ide o to aby sa zistilo, či bol softvér prívetivý alebo naopak, či bol

jednoducho ovládateľný, návodný, predvídateľný, zrozumiteľný a podobne. V oddelenej miestnosti sedí niekoľko ľudí, ktorí sa podieľajú na vývoji softvéru od projektového manažéra až po samotných testerov. Tí sa snažia zachytiť čo najviac poznatkov a majú za úlohu zistiť, čo je pre respondentov najobťažnejšie a akým spôsobom by sa to dalo vyriešiť. K zaznamenávaniu poznámok im postačí poznámkový blok prípadne dokument na počítači, pretože všetko je nahrávané kamerami a asistent moderátor taktiež zaznamenáva potrebné údaje.

5.1.6 Financie

Na to aby sa vedenie projektu alebo IT spoločnosti rozhodlo svoj produkt dôkladne testovať potrebuje pádny dôvod. Môže nastať situácia kedy pri menších IT projektoch nie je potrebné testovať vôbec prípadne použiť len niektoré z testovacích metód. Avšak z môjho pohľadu je lepšie investovať do testovania a vyhnúť sa riziku napriek o niečo nižším ziskom, ako toto riziko podstúpiť.

Pripomeňme si problém zvaný Y2K (Problém roku 2000). Tento problém bol výsledkom programovania základných funkcií softvéru a hardvéru, kedy sa vývojári snažili ušetriť počítačovú pamäť a tak bol rok zapisovaný len pomocou dvoch čísel. A preto bol rok 2000 interpretovaný nesprávne a to ako rok 1900. Ak by v dobe vývoja týchto funkcií bol softvér či hardvér riadne otestovaný, k tejto chybe by nemusel nikdy dôjsť. I keď následky neboli až tak fatálne, keďže verejný sektor vynaložil veľké finančné prostriedky na prevenciu prípadných komplikácií, v druhej polovici 90tych to vyvolalo obrovské obavy a úprava stála nemalé financie.

V projekte č.1 môžeme náklady na zaisťovanie kvality rozdeliť do dvoch kategórií a to na pravidelnú investíciu, ktorá zahŕňa cenu testovacieho tímu v určitom období, a nepravidelnú investíciu, ktorá sa týka hlavne nezávislého testovania. Toto testovanie prebehlo len dvakrát a predpokladá sa, že sa už nebude opakovať, jedine v prípade kedy by nastali určité nezhody a bolo by nutné opäť overiť užívateľskú prívetivosť softvéru. Každá z týchto kategórií je vyčíslená v nasledujúcich tabuľkách.

Tabuľka 2 Pravidelné investície vynaložené na testovanie

Pravidelná investícia				
Pozícia	**Počet pracovníkov**	**Cena za 1 mesiac brutto[5]**	**Cena za 6 mesiacov brutto**	**Cena za rok bruto**
Integračný tester	2	80 000Kč	480 000Kč	960 000Kč
UAT tester[6]	3	30 000Kč	180 000Kč	360 000Kč
Test manager	1	45 000Kč	270 000Kč	540 000Kč
Špecialista platobných kariet	1	40 000Kč	240 000Kč	480 000Kč
Spolu	7	195 000Kč	1 170 000Kč	2 340 000Kč

Zdroj: vlastné spracovanie

Tabuľka 3 Nepravidelné investície vynaložené na testovanie

Nepravidelná investícia				
Metóda testovania	**Účastníci**	**Počet účastníkov**	**Cena za jedno testovanie[7] brutto***	**Cena celkom brutto[8]**
Nezávislé testovanie	Integračný tester	2	4 000 Kč	8 000 Kč
	UAT tester	3	3 000 Kč	6 000 Kč
	Test manager	1	2 250 Kč	4 500 Kč
	Špecialista platobných kariet	1	2 000 Kč	4 000 Kč
	Moderátor	1	4 000 Kč	8 000 Kč
	Asistent	1	3 000 Kč	6 000 Kč
	Respondent	12	1 800 Kč	3 600 Kč
	Spolu	21	20 050 Kč	40 100 Kč

Zdroj: vlastné spracovanie

[5] Ceny sú približné odvodené z priemerného zárobku na obdobných pozíciách
[6] UAT testeri sú zamestnaný na polovičný úväzok
[7] Jedným testovaním sa myslia dva človekodni (16 hodín)
[8] Ceny sú konečné vrátane všetkých osobných poplatkov

Projekt začal v roku 2014 avšak testovanie naplno začalo až od septembra 2015. Predpokladané ukončenie projektu je koniec roku 2017. Testovanie na tomto projekte by teda celkovo malo trvať 28 mesiacov, čo by podľa predpokladaných nákladov stálo 5 500 100 Kč.

V prípade, že by sa projekt rozhodol vynechať testovanie kvôli šetreniu financií, mohlo by dôjsť k nasledujúcim chybám:

- **Zlá integrácia:**

Softvér na zariadení nefunguje vôbec, alebo funguje nesprávne. V prípade nesprávneho fungovania môžu byť následky skutočne zlé, keďže sa môže stať, že softvér nebude správne prepojený s pokladňou a tak sa môžu vyskytnúť chyby v transakciách či dotáciách. To by mohlo spôsobiť veľké straty spoločnosti a nespokojnosť klientov či naopak. Nefunkčnosť dotácií by znemožnila použiteľnosť zariadenia.

Rozsah škody pri zlej integrácii v prípade, že by zariadenie nefungovalo vôbec by bol len v strate zárobku na výberoch peňazí a zaplatení špecialistu, ktorý by problém vyriešil. Výška škody by sa odvíjala od toho ako dlho by oprava trvala. Avšak ak by systém nezaznamenával transakcie, prípadne by zaznamenával len niektoré, rozsah škody by sa mohol vyšplhať až do rádov miliónov korún v závislosti na tom ako dlho by tento stav trvala a koľko by za ten čas prebehlo transakcií.

Zlá integrácia teda môže predstavovať skutočnú hrozbu, avšak treba poznamenať, že pravdepodobnosť výskytu takýchto fatálnych chýb nie je vysoká. Počas môjho šesť mesačného testovania na projekte sme sa s podobnou chybou nestretli.

- **Chybná funkčnosť:**

V prípade chybnej funkčnosti ide opäť o veľmi širokú škálu problémov, ktoré môžu nastať. Chybnou funkčnosťou môžeme myslieť napríklad situáciu, kedy bankomat pri vložení platobnej karty hlási nedostatok finančných prostriedkov, napriek tomu, že je dostatočne nadotovaný. V tomto prípade, by škoda nebola až tak vysoká, pretože by šlo len o stratenie ziskov z uskutočnených výberov, prípadne by si dodávateľ služby mohol poškodiť meno u zákazníka, ktorý by si prostredníctvom bankomatu nemohol vybrať peniaze.

Iným prípadom chybnej funkčnosti však môže byť problém s vydávaním hotovosti. V tomto prípade sa jedná častejšie o hardvérovú ako softvérovú, chybu avšak

pri testovaní softvéru je ľahko odhaliteľná a rýchlo opravená. Naopak bez dostatočného otestovania môže napáchať vysoké škody. V prípade, že má bankomat chybný čítač prípadne chybné váženie bankoviek, je možné, že užívateľovi vydá nesprávnu hotovosť. Výška škody v tejto situácii opäť záleží na dobe do odhalenia chyby a preto sa môže vyšplhať do veľmi vysokej sumy.

Chybná funkčnosť sa taktiež objavila v takmer každej verzii, avšak počet týchto chýb prudko klesal. Nové chyby sa začali vyskytovať len v prípade, že zákazník zmenil svoj názor na niektoré časti softvéru a preto bolo nutné niektoré funkcionality zmeniť.

- **Chybné užívateľské rozhranie**

Chybným užívateľský rozhraním sa myslia chyby v podobe nesprávneho textu – zlá formulácia viet, gramatické chyby, ťažko pochopiteľný text, nesprávne zalamovanie textu, prekrytie textu a podobne. Taktiež to môžu byť chyby v grafickej podobe ako napríklad nevhodný obrázok alebo jeho nesprávne umiestnenie. Tieto chyby nemajú až tak veľký dopad na financie, ale na spokojnosť konečných užívateľov. Kvalita poskytovaných služieb sa znižuje a to môže v dlhodobom horizonte znamenať stratu zákazníka.

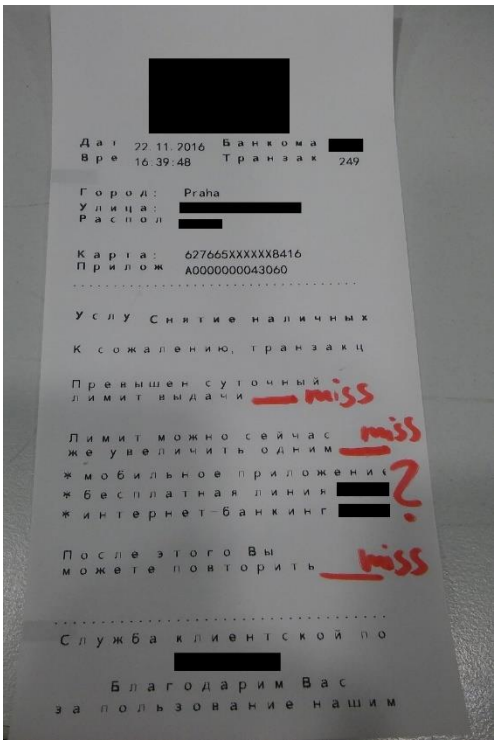
Chyby v užívateľskom rozhraní sa vyskytli v každej novej verzii softvéru. Je to hlavne z toho dôvodu, že pri úpravách sa nedáva až taký veľký dôraz na nezhody v texte ako napríklad rozdiel medzi tlačidlom „Ďalší“ a „Ďalej“, čo môžeme označiť ako minoritný problém. Keďže medzi jednotlivými verziami nie je toľko času, vývojári uprednostňujú opravu fatálnych chýb a preto sa minority „prenášajú“ z verzie na verziu.

- **Chyby v cudzích jazykoch**

Preklady zabezpečujú špecializované prekladateľské spoločnosti, avšak pri používaní väčšieho počtu jazykov sa môže stať, že sú jazyky implementované nesprávne. Existujú situácie kedy napríklad transakcie v anglickom jazyku vypíšu chybovú hlášku v českom jazyku. Najväčší problém býva s jazykmi ako sú ruština, čínština či japončina. Keďže tieto jazyky používajú iný druh písma veľmi často sa stáva, že majú nesprávne potvrdenku a podobne.

Chyby v cudzích jazykoch pretrvávali vo viac ako 50% verzií až kým sa softvér neprispôsobil cudzím znakom ako sú používané v ruskom či čínskom jazyku. Potom sa výskyt chýb výrazne znížil.

Obrázok 11 Chyba v ruskom jazyku na potvrdenke



Zdroj: Autor

Nie je možné presne odhadnúť, ktoré chyby nastanú a aký budú mať dopad. Na základe skúseností sa však dá predpokladať, ktorý typ chýb sa vyskytne s takmer sto percentnou istotou. Najčastejším druhom chýb sú chyby v užívateľskom rozhraní, potom je to chybná funkčnosť, ďalej chyby v prekladoch alebo problémy s cudzím jazykom a najmenej časté sú chyby v integrácii. Samozrejme sa môžu vyskytnúť aj iné problémy, avšak tieto štyri kategórie môžeme považovať za najfrekventovanejšie. Ak by sme mali vytvoriť percentuálny predpoklad výskytu chýb v jednotlivých kategóriách v tomto projekte, tak by to bolo:

- Zlá integrácia – 15%
- Chybná funkčnosť – 80%
- Chybné užívateľské rozhranie – 99%
- Chyby v cudzích jazykoch – 60%

Tieto predpoklady sú len približné, vytvorené na základe skúseností v období šiestich mesiacov.

Tabuľka 4 Náklady spôsobené chybným softvérom

Náklady na opravu chyby	**Doba opravy**			
	Deň	*Týždeň*	*Mesiac*	*Rok*
Vývojárske služby brutto[9]	8 000 Kč	40 000 Kč	160 000 Kč	1 920 000 Kč
Ďalšie náklady	**Strata zákazníka**	**Straty behom transakcií**	**Nepoužívaný softvér**	**Spolu**
Minimálna strata na zákazníka mesačne[10]	132 Kč	Od 0 Kč	32 Kč (1 výber = 8 Kč)	Min. 164 Kč
Druh chyby	*Predpoklad výskytu ďalších nákladov*			**Spolu na zákazníka**
Zlá integrácia		**Áno**	**Áno**	Min. 32 Kč
Chybná funkčnosť	**Áno**	**Áno**	**Áno**	Min. 164 Kč
Chybné užívateľské rozhranie	**Áno**		**Áno**	164 Kč
Chyby v cudzích jazykoch			**Áno**	32 Kč

Zdroj: Autor

Štatistika[11] transakcií na bankomatoch v Českej republike pre jednu spoločnosť

- Okolo 1 300 výberových bankomatov a 150 vkladových bankomatov
- Okolo 90 000 000 výberov ročne
- Okolo 35 000 000 dotazov na zostatok na účte ročne
- Okolo 4 300 000 platobných príkazov ročne
- Najpoužívanejší bankomat okolo 1 100 transakcií za deň
- Mesačný priemer výberov na bankomate je okolo 5 200
- Mesačný priemer všetkých transakcií na bankomate je okolo 7 720

Presnejšia štatistika vyhodnotená dňa 24.02.2017 na 64 bankomatoch

- 5 900 výberov denne, čo je približne 92 výberov na jeden bankomat
- 2 500 dotazov na zostatok
- 112 dobití mobilu

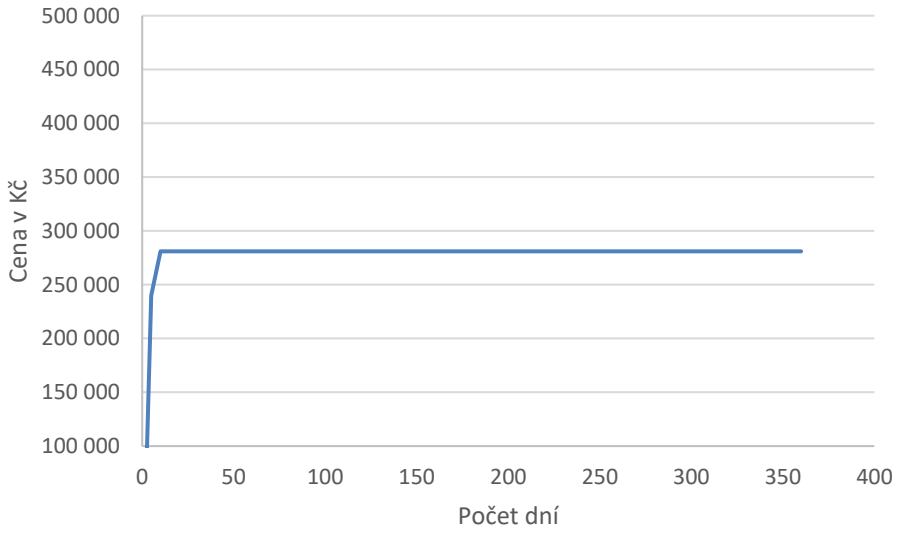
Priemerný počet nájdených defektov v jednej verzii nezávislé na druhu chyby je **30**.

[9] Cena za 1 hodinu činí 1000Kč
[10] Predpokladá sa, že zákazník využíva základ služby a nemá žiadne úvery
[11] Uvedené hodnoty sú približné a zaokrúhlené

- **Strata v najlepšom prípade**

Ak by oprava každej chyby trvala len jeden deň, stálo by to 240 000 Kč. Každý jeden deň opravy by spôsobil nedostupnosť softvéru. V najlepšom prípade počítame s tým, že by bol softvér použitý len na jedinom bankomate. Keďže priemerný počet výberových transakcií na bankomate za mesiac je 5 200, môžeme predpokladať minimálnu stratu 41 600 Kč. Ak by sa teda nevyskytli žiadne ďalšie náklady, minimálna strata by bola celkom 281 600 Kč a ďalej by sa nezvyšovala.

Obrázok 12 Konštantná strata v najlepšom prípade



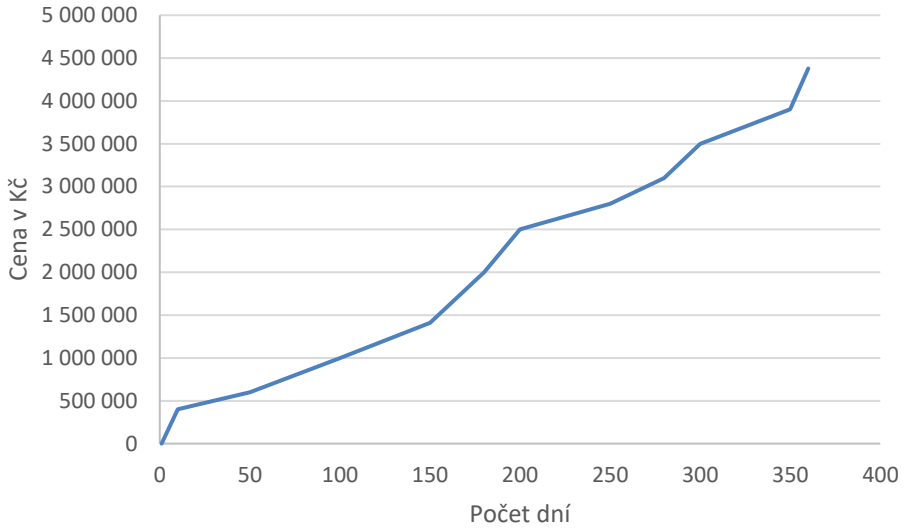
Zdroj: Autor

- **Strata v priemernom prípade**

Ak by oprava každej chyby trvala jeden týždeň, znamenalo by to 150 dní opráv a približné náklady by boli 1 200 000 Kč. V priemernom prípade počítame s tým, že kvôli nefunkčnému softvéru môžeme stratiť zákazníka. Jeden zákazník vykoná v priemere 4 výberové transakcie mesačne a preto môžeme odhadnúť z ročnej štatistiky približne 1 875 000 zákazníkov, ktorý za rok použili náš softvér. Pri strate 0,1% by šlo o zníženie počtu zákazníkov približne o 1 875 čo by znamenalo mesačnú stratu vo výške 247 500 Kč. Nedostupnosť softvéru by sa navýšila o počet dní opráv, čo by bolo približne 5 mesiacov a to by spôsobilo v prípade použitia softvéru len na jednom bankomate, škodu vo výške približne 208 000 Kč. Ak by sa nevyskytli žiadne ďalšie náklady, minimálna strata v priemernom prípade by bola celkom 1 408 000 Kč + každý mesiac náklady na

stratených zákazníkov čo by bolo za rok 2 970 000 Kč. Celkovo by teda za rok strata predstavovala čiastku 4 378 000 Kč.

Obrázok 13 Rastúca strata v priemernom prípade



Zdroj: Autor

- **Strata v najhoršom prípade**

Keďže sa v prípade chybnej integrácie môže strata vyšplhať nekonečne vysoko, preto aj stratu v najhoršom prípade môžeme považovať za nekonečne vysokú.

5.1.6.1 Vyhodnotenie financií

Je potrebné si uvedomiť, že pravdepodobnosť prvého prípadu je takmer nulová, keďže oprava fatálnych chýb môže trvať častokrát dlhšie ako týždeň. Najpravdepodobnejšie teda je, že sa ocitneme v priemernom prípade. Tu sa náklady takmer rovnajú výdavkom vynaloženým na testovanie, ale stále sú nižšie. Treba si však uvedomiť, že existuje veľa faktorov, ktoré môžu situáciu zhoršiť. Častokrát sa pri oprave chyby stane, že sa vyskytne iná neočakávaná chyba a v tom prípade by sa celý proces opráv musel zopakovať čo by mohlo znamenať zdvojnásobenie nákladov a tým by náklady na opravu zjavne prevyšovali náklady na testovanie. Takáto situácia sa môže zopakovať niekoľkokrát a tiež musíme počítať so stratou v najhoršom prípade, ktorej pravdepodobnosť síce nie je vysoká, ale existuje. V prípade, že by bol softvér hneď na začiatku použitý na viac ako jednom bankomate, straty by boli niekoľkonásobne vyššie. A preto je testovanie veľmi dôležitou súčasťou vývoja softvéru.

5.2 Projekt č.2

Informácie o projekte číslo 2 neboli získané na základe osobnej skúsenosti, ale vďaka konzultácii s testovacím manažérom iného projektu. Ide o projekt, ktorý začal vytvárať otvorené REST API (Representational State Transfer application programming interface)[12] v korporáte. Prevažne je zameraný na internetové bankovníctvo. Projekt sa snaží vyvíjať pomocou agilnej metodiky scrum, avšak aplikácia tohto spôsobu vývoja je len na počiatku a preto metodika scrum nie je dodržiavaná úplne presne aj vzhľadom k tomu, že je projekt integrovaný na ostatné projekty v banke. Projekt sa skladá zo štyroch tímov, ktoré pracujú v štrnásť-denných šprintoch. Počas každého šprintu prebehne jedna retrospektíva a plánovanie a podľa potreby niekoľkokrát grooming.

Jednotkové testy v tomto projekte používajú vývojári, ktorí úzko spolupracujú s testovacím tímom. Tieto testy používajú predovšetkým ako kontrolu aby si po sebe mohli overiť svoj kód, či funguje správne prípadne či pridaním svojho kódu niečo nerozbili.

Integračné testy sa používajú len v prípade, pokiaľ je potrebné vyvíjať a používaný backend[13] ešte danú funkčnosť nemá. V tom prípade sa vytvorí potrebný kód oproti simulačnému režimu, ktorý zastupuje backend. Ide o takzvané mockované volanie čo znamená, že nezáleží na tom aký tvar má vstup, pretože výstup bude vždy rovnaký. Ihneď po tom ako má backend svoju funkčnosť hotovú sa priamo naň napojí a integračne sa testuje, či všetko funguje rovnako ako to bolo proti mocku.

Systémové testy sú klasickými testami, ktoré sa používajú pri vývoji. Pomocou týchto testov sa overuje vytvorená funkčnosť oproti dokumentácii. Pre tento druh testovania sú na projekte používane nástroje pre restové volania a to konkrétne Postman a Insomnia. Po tom sa testy pridelia do runscope pre automatizované testy.

Akceptačné testy sa tu nerobia v úplne pravom slova-zmysle, ale jednoducho sa pre každý prípad vytvoria akceptačné kritériá, ktoré musia po vývoji prejsť testami aby sa prípad mohol uzavrieť ako dokončený.

Nezávislé testovanie na tomto projekte neprebieha. Je to z toho dôvodu, že projekt je zameraný na backend a nezávislé testovanie sa používa hlavne pri testovaní frontendu[14].

[12] REST API: architektúra rozhraní, navrhnutá pre distribuované prostredie
[13] Backend: zahŕňa štruktúru zdrojového kódu, neviditeľné priebehy skriptov
[14] Frontend: predstavuje viditeľný obsah softvéru

5.2.1 Náklady projektu č.2

Na projekte pracuje testovací tím, ktorý sa skladá z testovacieho manažéra a testerov. Dvaja testeri pracujú na plný úväzok a ďalší štyria na skrátený úväzok. Na rozdiel od predchádzajúceho projektu tu nie sú žiadne jednorazové náklady pretože testovanie prebieha v jednej línii vždy za rovnakých podmienok.

Tabuľka 5 Náklady na testovanie softvéru

Náklady na testovanie			
Pozícia	**Počet pracovníkov**	**Cena za 1 mesiac brutto**[15]	**Cena za rok bruto**
Hlavný tester	2	80 000Kč	960 000Kč
Tester	4	40 000Kč	480 000Kč
Test manager	1	45 000Kč	540 000Kč
Spolu	7	165 000Kč	1 980 000Kč

Zdroj: Autor

Náklady na testovanie sú po celú dobu projektu stále. Predpoklad dĺžky testovacej fázy je odhadnutý na štyri roky. Keďže sa projekt týka tej istej spoločnosti ako v predchádzajúcom prípade, hrozby pri nezaistení dostatočnej kvality sú rovnaké ako v projekte č.1. V priemernom prípade by sa teda projekt ocitol v strate 2 970 000 Kč, čo výrazne presahuje investíciu do testovania. Z toho vyplýva, že investícia do testovania má rozhodne veľký význam aj v tomto projekte.

[15] Ceny sú približné odvodené z priemerného zárobku na obdobných pozíciách

Záver

Zabezpečenie kvality softvéru zahŕňa overovanie naplnenia požiadaviek zákazníka, kontrolu zhody so zadaním a špecifikáciou projektu, kontrolu výstupov vývojárov zameranú na dodržiavanie vopred definovaných štandardov a noriem a testovanie za účelom hľadania chýb - funkčných, logických, obsahových, systémových.

Pre porovnanie kvality softvéru existuje niekoľko štandardov, metrík a spôsobov merania kvality. Najznámejší je štandard ISO, ktorý však nie je o kvalite výrobkov, ale o kvalite celého výrobného procesu, kvalite manažmentu a kvalite dokumentácie. V oblasti kvality sa však stále vykonáva výskum, ktorý určite zabezpečí svetlejšiu budúcnosť v oblasti kvality softvérových aplikácií. Výsledky síce nemožno čakať ihneď, ale časom sa určite do oblasti kvality zavedú pevné a spoľahlivé štandardy.

Pod slovom kvalita si každý predstaví niečo mierne odlišné. Snaha definovať kvalitu viedla k vzniku rôznym modelom kvality. Podľa normy ISO/IEC 25010 je kvalita mierou splnenia stanovených potrieb, ak je produkt používaný za stanovených podmienok. Model FURPS je výsledkom prístupu k definícii a merania jednotlivých atribútov kvality dodaného systému.

Cena za kvalitu nie je malá a nie je jednoduché stanoviť presnú hranicu medzi kvalitou a nákladmi na ňu. Snahou v každej spoločnosti je vždy s minimálnymi nákladmi dosiahnuť maximálnu kvalitu. To však vôbec nie je jednoduché, dá sa povedať, že v podstate nemožné. Minimalizovanie nákladov na vývoj softvéru so sebou vždy prináša veľké riziká čo sa na kvalite produktu môže výrazne prejaviť.

Môžeme tvrdiť, že tu platí jednoduché pravidlo: Čím skôr je chyba odhalená, tým lacnejšie je jej odstránenie. Avšak na to aby sme boli schopní chyby opraviť čo najskôr, potrebujeme zabezpečiť častejšie a dôkladnejšie testovanie, ktoré sa nám na nákladoch samozrejme prejaví ako zvýšenie. Náklady na kvalitu môžeme rozdeliť na náklady vynaložené na prevenciu, na preverenie a na detekciu chýb. Do prevencie patrí plánovanie akosti, školenie tímu a tiež zabezpečenie nástrojov na testovanie. Náklady na preverovanie zahŕňajú cenu za preskúmanie, inšpekcie a testovanie. Náklady na odstránenie chyby sa líšia, ak bola chyba objavená pred dodaním zákazníkovi (vnútorná chyba) alebo po dodaní (vonkajšia chyba). Ukazuje sa, že náklady dramaticky rastú od prevencie defektov k ich detekcii, od odstraňovania vnútorných chýb

k odstraňovaniu chýb vonkajších. Napriek tomu prieskum v oblasti reálnych IT projektoch potvrdil, že investícia do zabezpečenia kvality softvéru je skutočne dôležitá a jej ignorovanie môže spôsobiť obrovské straty. V prípade malých projektov a drobných zmien môže byť táto investícia otázna, to sa však týka len skutočne projektov veľmi malého rozsahu kde nenachádzame takmer žiadne hrozby.

Overovanie kvality, zaisťovanie kvality a celkové testovanie počas celého vývoja produktu v podobe softvéru jednoznačne odporúčam, pretože investície nikdy nebudú tak nákladné aké by boli škody, ktoré by mohli nastať. Pokiaľ si chce spoločnosť udržať dobré meno a zákazníkov, musí ponúkať kvalitné produkty a to dosiahne jedine starostlivým overovaním kvality a používaním správnych testovacích metodík. Testovanie je vhodné nie len pre veľké, ale tiež aj pre malé projekty pri vhodnej testovacej stratégii. Meranie kvality by som odporučila v každom projekte bez ohľadu na zameranie a veľkosť, pretože tieto ukazatele nám dokážu podať informáciu o aktuálnom stave projektu, na základe čoho sme schopný dedukovať jeho budúci priebeh a zvoliť tak ďalší postup správne. Výsledky metrík dokážu odhaliť skryté nedostatky projektu a vyvarovať sa tak nečakanému krachu.

Použité zdroje

Literatúra

[1] *Řízení kvality softwaru*. Petr ROUDENSKÝ, Anna HAVLÍČKOVÁ, COMPUTER PRESS, 2013, 9788025138168

[2] *Handbook od Usability Testing*, Jeffrey Rubin, Dana Chisnell, Wiley, 2008, 978-0-470-18548-3

[3] *Andrew Hunt and David Thomas. Pragmatic Unit testing*. The Pragmatic Programmers. ISBN-10: 0974514012, 2001 pages 1-10; 65-70.

[4] *Is ISO 9000 really a standard?* Jim Wade. ISO Management Systems (May-June 2002).

Internetové zdroje

[5] Joel E. ROSS. *Total Quality Management*. Dostupné z: https://totalqualitymanagement.wordpress.com/2009/08/27/definition-of-quality/

[6] Doc. Ing. Pavol TANUŠKA, PhD., Doc. Ing. Peter SCHREIBER, CSc.. *Proces testovania softvérových produktov.* Dostupné z: https://www.mtf.stuba.sk/docs//internetovy_casopis/2005/5/tanuska.pdf

[7] ALGOMICA SOFTWARE. *Vývoj a testování softwaru*. Dostupné z: http://www.algomica.cz/testovani_uvod.htm

[8] WEB-INTEGRATION. Manažment kvality a testovanie softvéru ako súčasť webovej integrácie. Dostupné z: http://www.web-integration.info/cs/blog/manazment-kvality-a-testovanie-softveru- ako-sucast-webovej- integracie

[9] QUALIDADEBR. *FURPS+*. Dostupné z: https://qualidadebr.wordpress.com/2008/07/10/furps/

[10] WIKIPEDIA. *FURPS*. Dostupné z: https://cs.wikipedia.org/wiki/FURPS

[11] TECHNOLOGYUK. *The Waterfall Model.* Dostupné z: https://www.technologyuk.net/computing/sad/waterfall-model.shtml

[12] Jack BARBER. *The V Model*. Dosupné z: https://buildingstrongerpartners.wordpress.com/tag/v-model/

[13] PRINTEREST. *ALM*. Dostupné z: https://cz.pinterest.com/pin/535858055642470880/

[14] MICROSOFT. *Projektový trojuholník*. Dostupné z: https://support.office.com/sk-sk/article/Projektov%C3%BD-trojuholn%C3%ADk-8c892e06-d761-4d40-8e1f-17b33fdcf810

[15] MANAGEMENT MANIA. *Magický trojuholník projektového riadenia*. Dostupné z: https://managementmania.com/sk/magicky-trojuholnik-projektoveho-riadenia

Ostatné zdroje:

[16] Peter ZOŠIAK. *Principy a postupy vytváření automatizovaných testů v MS Visual Studio 2012*. Praha, 2013/2014. Diplomová práca. Vysoká škola finanční a správní v Praze, o.p.s. Vedúci práce RNDr. Jan Lánský, Ph.D.

[17] Bc. Martin SOKOL. *Automatické testování webových aplikací*. Brno, 2013. Diplomová práca. Masarykova univerzita Fakulta informatiky. Vedúci práce RNDr. Radek Ošlejšek, PhD.