EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/360861297699095O8

Návrh a implentácia nástroja na vyhodnocovanie testov

Diplomová práca

Bc. Rita Grosschmidtová

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Návrh a implementácia nástroja na vyhodnocovanie testov

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Kvantitatívne metódy v ekonómii

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: Ing. Igor Bandurič Phd.

Bratislava 2018 Bc. Rita Grosschmidtová

Čestné vyhlásenie

Čestne vyhlasuje, že záverečnú prácu som vypracovala samostatne a že som uviedla všetku použitú literatúru.

Dátum:

Poďakovanie

Chcela by som poďakovať všetkým, ktorí mi pomáhali počas písania záverečnej práce. Predovšetkým ďakujem môjmu školiteľovi Ing. Igorovi Banduričovi, PhD. za ochotný prístup, pomoc pri písaní práce a aj za poskytnutie cenných rád.

Abstrakt

GROSSCHMITOVÁ, Rita: *Návrh a implementácia nástroja na vyhodnocovanie testov.* – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Ing. Igor Bandurič, Phd.- Bratislava: FHI, 2018, 69 s.

Cieľom záverečnej práce je objektovo orientovaný návrh a implementácia nástroja, ktorý umožní vyhodnocovať stav testovania softvéru. Návrh nástroja bude vychádzať z nedostatkov (vrátane prílišnej zložitosti) existujúcich nástrojov na slovenskom a zahraničnom trhu. Práca je rozdelená do štyroch kapitol. Obsahuje 2 tabuľky, 25 obrázkov a 1 prílohu. Prvá kapitola je venovaná definícii testovania a súčasnej ponuke možnosti vyhodnocovania testovania pomocou existujúcich nástrojov, druhá kapitola obsahuje popis technológií, ktoré boli použité pri analýze a programovaní webovej aplikácie. V tretej kapitole opisujeme cieľ našej diplomovej práce, pričom definujeme aj čiastkové ciele práce. Záverečná kapitola obsahuje výsledky diplomovej práce. Skladá sa z troch hlavných častí – požiadavky na informačný systém, návrh informačného systému a implementácia informačného systému. V prvej časti definujeme funkčné a nefunkčné požiadavky na systém, pomocou ktorých navrhujeme nástroj na vyhodnocovanie testov, ktorý následne implementuje vo vybranom jazyku a prostredí. Výsledkom riešenia danej problematiky je návrh nástroja na vyhodnocovanie testov a implementácia jeho časti.

Kľúčové slová: webová aplikácia, testovanie, ASP. NET MVC, UML, OO návrh

Abstract

GROSSCHMIDTOVÁ, Rita: *Design and implementation of test evaluation tool.* – University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. – Ing. Igor Bandurič, PhD. – Bratislava: FHI, 2018, 69 p.

The aim of our final thesis is object-oriented design and implementation of test evaluation tool. Design of the tool will be based on disadvantages (including the excessive complexity) of existing instruments on the Slovak and foreign markets. The thesis is divided into four chapters. The thesis contains 2 tables, 25 images and 1 attachment. The first chapter is dedicated to the definition of testing itself and the current offer of existing evaluation testing tools, the second chapter contains a description of the technologies that will be used to analyze and program the web application. In the third chapter we will describe the goal of our final thesis, while defining the partial aims of the thesis. The final chapter contains the results of the thesis. It consists of three main parts – requirements of information system, information system design and implementation of the designed tool. In the first part, we will define functional and non-functional requirements to design a test evaluation tool which we will program in the selected programming language and framework. The solution to this problem is the design of the tool for the evaluation of the testing and the implementation part of this design.

Key words: web application, testing, ASP. NET MVC, UML, object-oriented design

Obsah

Zoznam ilustrácií a zoznam tabuliek

Úvod

V súčasnosti softvérové aplikácie majú veľký vplyv na obchodný výsledok firiem. V konkurenčnom boji sú spoločnosti tlačené do najvyššej možnej miery finančnej efektivity a flexibility. Preto vyžadujú čo najrýchlejšie dodanie softvérových aplikácií a primeraný pomer nákladov a výnosov. Z toho dôvodu je vývoj softvérových aplikácií pod tlakom a je potrebné aby dané riešenie bolo dodané čo najrýchlejšie za čo najnižšiu cenu, ale v primeranej kvalite. S použitím moderných technológií nároky na kvalitu dodávaného softvéru stúpajú a práve testovanie zastrešuje kontrolu kvality dodávaného softvéru.

Diplomová práca sa v teoretickej časti zaoberá samotnou definíciou testovania, procesom testovania a definovaním jednotlivých modelov testovania, aby sme pochopili jeho význam a dôvod, prečo je dôležitou súčasťou vývojového cyklu softvérového produktu. Taktiež sme sa v teoretickej časti našej diplomovej práce zaoberali súčasným stavom danej problematiky doma i v zahraničí. Ako príklad sme vybrali dva komerčné testovacie nástroje, ktoré sú využívané spoločnosťami po celom svete, opísali sme ich funkcionality, výhody a aj nevýhody.

Medzi najväčšie nevýhody patrí cena, ktorú spoločnosť musí investovať dokúpením nadstavby systému, ktorá by podporovala priamo vyhodnocovanie testovania daného projektu v systéme. Najmä kvôli privysokej cene sú nútení test a projektoví manažéri uchovávať dáta o stave testovanie a projektu mimo softvéru, napríklad vo forme excelovských súborov. Na základe zistených nedostatkov daných nástrojov sme navrhli v praktickej časti našej diplomovej práce nástroj, ktorý bude v budúcnosti slúžiť ako voľne dostupný doplnok, ktorý by si napájal na databázy už existujúcich softvérov. Nami navrhovaný program by umožnil prístup k základným a súčasnej aj detailnejším informáciám o stave projektu a testovania nielen pomocou rôznych typov náhľadov ale aj pomocou metrík určených na vyhodnotenie stavu testovania softvérového produktu.

1 Súčasný stav riešenej problematiky doma a v zahraničí

1.1 Definícia testovania

Testovanie môžeme definovať ako spúšťanie testov. Avšak ide len o súčasť testovania. Testovacie aktivity existujú pred a po vykonaní testov. Medzi tieto aktivity zahŕňame plánovanie, riadenie, výber testovacích podmienok, navrhovanie a vykonávanie testovacích prípadov, kontrolu výsledkov, vyhodnocovanie výstupných kritérií, reportovanie testovacieho procesu a testovaného systému a finalizáciu ukončovacích aktivít po testovaní. [1]

„Testovanie môže mať nasledujúce ciele:

- Nájdenie defektov
- Získanie dôvery s ohľadom na úroveň kvality
- Poskytnutie informácií pre rozhodovanie
- Predchádzanie defektom“[1]

Podľa učebnej osnovy pre základný stupeň certifikovaného testera môžeme definovať niekoľko princípov testovania, ktoré poskytujú všeobecné pokyny spoločné pre každé testovanie, ktoré boli navrhnuté za posledných 40 rokov.

„1. Princíp – Testovanie ukazuje prítomnosť defektov

Testovanie môže ukázať, že sú defekty prítomné, ale nemôže dokázať, že v softvéri nie sú žiadne defekty. Testovanie znižuje pravdepodobnosť, že v softvéri zostanú neobjavené defekty, avšak nenájdenie žiadneho defektu stále nie je dôkaz jeho bezchybnosti.

2. Princíp – Vyčerpávajúce testovanie je nemožné

Testovanie všetkého (všetkých kombinácií vstupov a vstupných podmienok) nie je realizovateľné s výnimkou triviálnych prípadov. Namiesto vyčerpávajúceho testovania by mala byť k určeniu hlavného predmetu testovacieho úsilia použitá analýza rizík a stanovenie priorít.

[1]CASTB – *Učebná osnova pre základný stupeň*, 2013 [Online]. [cit. 2017-12-11] Dostupné na internete: <http://castb.org/wp-content/uploads/2013/11/ISTQB_CT_Zakladny_stupen_v2011_SK_Beta2.pdf>

3. Princíp – Včasné testovanie

Pre včasnú identifikáciu defektov, musia testovacie aktivity začať v rámci životného cyklu vývoja softvéru alebo systému tak skoro, ako je to len možné a musia byť zamerané na definované ciele.

4. Princíp – Zhlukovanie defektov

Testovacie úsilie musí byť zamerané proporčne na očakávanú a neskôr zistenú hustotu defektov v moduloch. Malé množstvo modulov zvyčajne obsahuje väčšinu defektov zistených počas testovania pre uvoľnením, alebo je zodpovedné za najviac prevádzkových zlyhaní.

5. Princíp – Pesticídny paradox

Ak sú tie isté testy opakované, časom ten istý súbor testovacích prípadov nenájde žiadne nové defekty. Na prekonanie tohto paradoxu je potrebné existujúce testovacie prípady pravidelne revidovať a upravovať a je potrebné napísať nové, odlišné testy na vykonanie iných častí softvéru alebo systému pre prípadné odhalenie ďalších defektov.

6. Princíp – Testovanie je závislé na kontexte

Testovanie je vykonávané odlišne v rôznych kontextoch. Napríklad softvér kritický z pohľadu bezpečnosti sa testuje iným spôsobom ako webové stránky elektronického obchodu.

7. Princíp – Falošná predstava o neexistencii omylov

Nájdenie a opravenie defektov nepomôže, ak je vytvorený systém nepoužiteľný a nespĺňa potreby a očakávania užívateľov.“[2]

1.2 Proces testovania softvéru

Trend testovania sa v dnešnej dobe mení. Od testera požadujeme aby bol viac technicky a procesne orientovaný. Testovanie nemôžeme považovať ako proces iba na

[2]CASTB . *Učebná osnova pre základný stupeň*. 2013 [Online]. [cit. 2018-01-26] Dostupné na internete: <http://castb.org/wp-content/uploads/2013/11/ISTQB_CT_Zakladny_stupen_v2011_SK_Beta2.pdf> (str. 14)

nájdenie defektov, ale má širší rozsah. Je potrebný od samotného začiatku projektu, ešte predtým než definujeme konečné požiadavky.

Testovanie môžeme definovať ako štandardizovaný proces. Z toho dôvodu má aj svoj životný cyklus. Životný cyklus testovania softvéru odkazuje na postupnosť zmien z jednej fázy na inú. Podobne ako softvér cyklus zahŕňa sekvencie krokov, ktoré by mali byť vykonané v určitom poradí.

Životný cyklus testovania sa vzťahuje na proces, ktorý má vykonávať konkrétne kroky v určitej postupnosti, aby sme zabezpečili splnenie kvalitatívnych cieľov. V procese STLC sa každá činnosť uskutočňuje plánovaným a systematickým spôsobom, pričom každá fáza má rôzne ciele a výsledky. STLC v každej organizácií obsahuje rôzne fázy, základ však ostáva rovnaký.

Medzi fázy testovacieho cyklu uvádzame:

1. Fáza požiadaviek – počas tejto fázy analyzujeme a študujeme požiadavky. Prebiehajú stretnutia s inými tímami, ktoré zisťujú či sú požiadavky testovateľné alebo nie. Táto fáza pomáha identifikovať rozsah testovania. Ak nejaká funkcia nie je testovateľná, je potrebné ju definovať počas fázy požiadaviek, aby mohla byť plánovaná tzv. stratégia zmierňovania.

2. Fáza plánovania – v niektorých scenároch považujeme plánovanie testov za prvý krok testovacieho procesu. V tejto fáze identifikujeme aktivity a zdroje, ktoré by pomohli splniť ciele testovania. Počas plánovania sa snažíme identifikovať aj metriky, metódu zhromažďovania a sledovania týchto metrík. Neplánujeme však len požiadavky, tie tvoria jeden zo základov. Existujú však ďalšie dve veľmi dôležité faktory, ktoré výrazne môžu ovplyvňovať plánovanie testov. Medzi tieto faktory zaraďujeme Testovaciu stratégiu organizácie a Analýzu a riadenie rizík.

3. Fáza analýzy – počas fázy analýzy definujeme predmet, ktorý sa má testovať. V podstate analyzujeme prostredníctvom dokumentácie požiadaviek, výrobných rizík a iného testovacieho základu podmienky testovania. Navrhnuté podmienky by mali byť

spätne vysledovateľné. Poznáme rôzne faktory, ktoré ovplyvňujú identifikáciu testovacích podmienok:

- úroveň a hĺbka testovania
- zložitosť softvérového výrobku
- riziká pre produkt a projekt
- životný cyklus vývoja softvéru
- manažment testov
- zručnosti a znalosti tímu
- dostupnosť zainteresovaných strán

Testovacie podmienky by sme mali napísať podrobným spôsobom, pretože zvyšuje pokrytie testu. Testovacie prípady sú potom napísané na základe testovacieho stavu, tieto podrobnosti povedú potom k napísaniu podrobnejších testovacích prípadov, ktoré nakoniec zvýšia pokrytie samotného testu. Taktiež je potrebné, aby sme počas fázy analýzy určili výstupné kritériá testovania, t.j. podmienky, kedy zastavíme proces testovania.

4. Fáza návrhu – fáza zahŕňa nasledujúce úlohy:

- podrobnosti o podmienkach testu – rozdelením testovacích podmienok do viacerých skupín čiastkových podmienok, aby sme zvýšili veľkosť pokrytia
- identifikácia a získanie testovacích dát
- identifikácia a nastavenie testovacieho prostredia
- vytvorenie kritérií sledovateľnosti požiadaviek
- vytvorenie metriky pokrytia testov

5. Implementačná fáza – hlavou úlohu tejto fázy je vytvorenie podrobných testovacích prípadov. Taktiež určujeme testovacie prípady, ktoré sa stanú súčasťou balíka na regresné testovanie. Pred dokončením testovacieho prípadu je dôležité, aby sme vykonali jeho revíziu. Ak náš projekt zahŕňa automatizované testy, je potrebné aby sme identifikovali aj kandidátov na automatizáciu a spracovali testovacie skripty.

6. Fáza realizácie – ako nám naznačuje názov fázy ide o časť životného cyklu, v ktorej sa uskutočňuje skutočná realizácia testov. Skôr ako začneme testy realizovať musíme sa uistiť, že sú splnené všetky vstupné kritériá. V tej fáze vykonávame testovanie podľa testovacích scenárov a zaznamenávame defekty v prípade akýchkoľvek nezrovnalostí. Taktiež sledujeme pomocou daných metrík pokrok v testovaní.

7. Záverečná fáza – vo fáze sa sústreďujeme na výstupné kritéria a na vykazovanie. V závislosti od nášho projektu a výberu účastníkov na ňom sa môžeme rozhodnúť, či chceme vytvárať denné alebo týždenné reporty. Existujú rôzne typy reportov, medzi ktoré patria napríklad DSR – Denný report o stave testovania a WSR - týždenný report o stave testovania. Je však dôležité spomenúť, že obsah správy jednotlivých reportov sa mení a závisí od toho, komu je správa adresovaná.

8. Ukončovacia fáza – medzi úlohy súvisiace s ukončovacou fázou:

- skontrolujeme kompletizáciu testov – či všetky testovacie prípady a scenáre boli vykonané
- skontrolujeme, či nie sú otvorené defekty najvyššej priority
- vypracujeme poznatky z testovania a vytvoríme dokument so získanými skúsenosťami z testovania, ktorý zahŕňa, čo počas testovania fungovalo dobre a kde je priestor na zlepšenie[3][]

[3]Softwaretestinghelp.com,2016. *What is software testing life cycle stlc*.[Online].[cit. 2017-11-30]. Dostupné na internete:< http://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/>

Názov fázy	Vstupné kritériá	Výstup
Fáza požiadaviek	Dokument špecifikácie požiadaviek	RUD (Dokument pochopenia požiadaviek)
	Dokument dizajnu aplikácia	Report o možnostiach realizácie testov
	Dokument používateľsky akceptačných kritérií	Report o možnostiach realizácie automatizovaných testov
Fáza plánovania	Aktualizovaný dokument špecifikácie požiadaviek	Dokument plánovania testov
	Report o možnostiach realizácie testov	Dokument o znižovaní rizika
	Report o možnostiach realizácie automatizovaných testov	Dokument odhadu testov
Fáza analýzy	Aktualizovaný dokument špecifikácie požiadaviek	Dokument podmienok testovania
	Dokument plánovania testov	
	Risk dokument	
	Dokument odhadu testov	
Fáza návrhu	Aktualizovaný dokument špecifikácie požiadaviek	Detailný dokument podmienok testovania
	Dokument podmienok testovania	Metriky sledovateľnosti požiadaviek
		Metriky pokrytia testov
Implementačná fáza	Detailný dokument podmienok testovania	Testovacie scenáre, skripty a testovacie dáta
Fáza vykonávania	Testovacie scenáre	Report vykonávania testov
		Report defektov
	Testovacie skripty	Záznam z testovania
		Aktualizované metriky sledovateľnosti požiadaviek
Záverečná fáza	Aktualizované testovacie scenáre s výsledkami	Aktualizované metriky sledovateľnosti testov
	Podmienky na uzavretie testov	Súhrnný report testovania
		Aktualizovaná správa o riadení rizika
Ukončovacia fáza	Podmienky na uzavretie testov	Správa o ukončení testu
	Report zhrnutia testov	

Tabuľka 1 Prehľad vstupov a výstupov v STLC [Zdroj: vlastné spracovanie]

Obrázok 1 Fázy testovacieho cyklu [Zdroj: vlastné spracovanie]

1.3 Procesné modely testovania softvéru

Cieľom procesných modelov pre proces vývoja softvérov je definovanie aktivity, ktoré je potrebné počas vývoja softvéru vykonať a súčasne definovať aj postupnosť ich vykonávania. Používaním procesných modelov sa výrazne zefektívnil proces vývoja softvéru a to najmä z hľadiska plánovania a riadenia zdrojov. Takéto procesné modely vieme určiť aj pre fázu testovania softvéru, ale sú menej známe, najmä preto, lebo testovanie považujeme za akúsi pomocnú alebo vedľajšiu činnosť, ktorá musí byť vykonávaná po ukončení implementácie. Ako sme však spomínali v úvode, testovanie prebieha už počas samotnej implementácie, nie po jej ukončení. Podľa niektorých publikácií testovanie tvorí približne 30 až 40% celkových nákladov softvérového projektu. V nasledujúcej kapitole sa budeme zaoberať stručným prehľadom procesných modelov na testovanie softvéru.

1.3.1 V-model

Model považujeme za jeden z najznámejších a najvyužívanejších modelov v praxi. Tento model nám ukazuje súvislosti medzi vývojovými aktivitami a testovacími aktivitami jednoduchým a zrozumiteľným spôsobom.

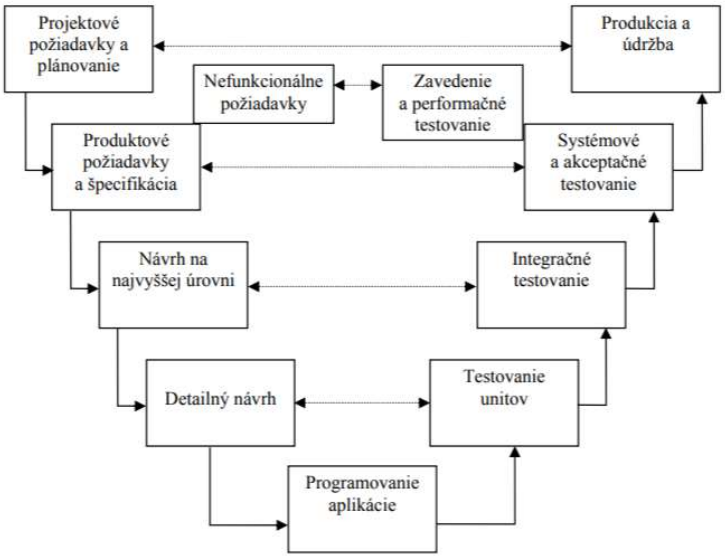
Skladá sa z dvoch vetiev, ktoré spoločne vytvárajú tvar písmena V. Ľavá vetva modelu nám zobrazuje vývojové aktivity a ich chronologickú postupnosť v procese vývoja softvéru a pravá vetva nám zobrazuje testovacie aktivity. Vodorovné šípky ukazujú, ktorá vývojová aktivita je podkladom pre danú testovaciu aktivitu na rovnakej úrovni.

Proces vývoja softvéru v ľavej vetne modelu sa začína špecifikáciou požiadaviek, potom pokračuje špecifikáciou funkcií systému, technickou špecifikáciou, programovou špecifikáciou a poslednou aktivitou je programovanie, resp. implementácia.

Aktivity testovacieho procesu rozdeľujeme v tomto modeli do štyroch základných úrovní:

- **Testovanie komponentov** – ide o testovanie funkčne ohraničených elementárnych jednotiek programu. Môžeme sem zaradiť napríklad funkciu, procedúru alebo triedu. Toto testovanie považujeme za testovanie softvéru na najnižšej úrovni.
- **Integračné testovanie** – integračné testovanie testuje schopnosť vzájomnej integrácie už otestovaných elementárnych jednotiek, prípadne ich skupín
- **Systémové testovanie** – v tomto prípade testujeme na úrovni celého systému. Na tvorbu testovacích prípadov využívame skutočné scenáre, ktorých vykonávanie očakávame od vyvíjaného systému
- **Akceptačné testovanie** – pri akceptačnom type testovaní pracujeme už s vyvinutým systémom, a to spôsobom, ako bol pre nás vyvinutý. Testovanie prebieha u dodávateľa ale aj u odberateľa systému

Z V-modelu vyplýva, ktorá vývojová aktivita produkuje podkladové materiály pre ktorú testovaciu aktivitu a v podstate vychádza z aktivít životného cyklu softvéru. [2]



Obr. 1. *V- procesný model*

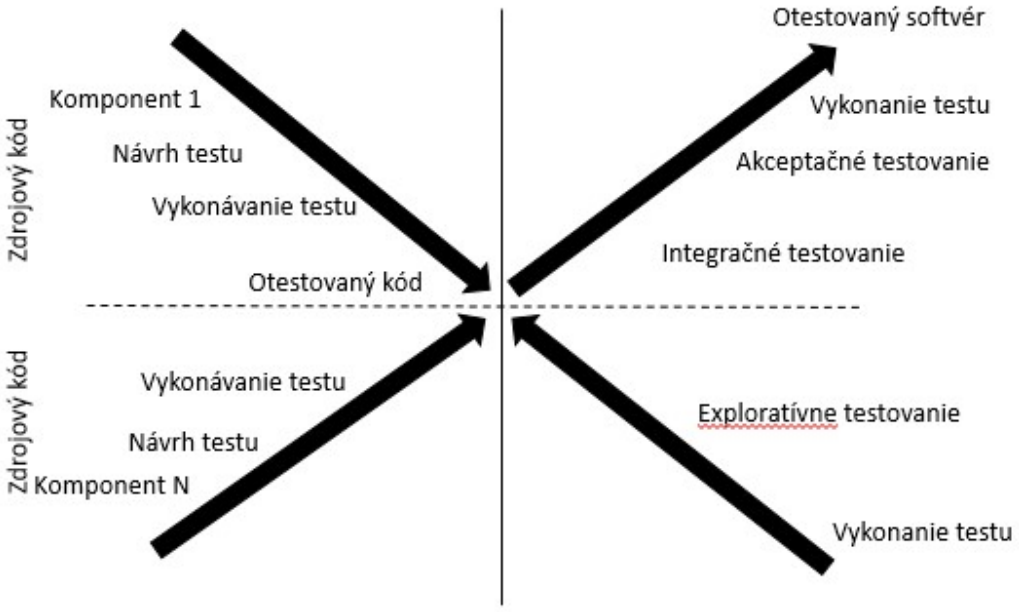
Obrázok 2 Zobrazenie V-modelu [Zdroj: www.mtf.stuba.sk]

1.3.2 X-model

X-model nepovažujeme za typ procesného modelu, ktorý sa zaoberá aktivitami životného cyklu softvéru. Na základe tohto modelu môžeme definovať základné testovacie aktivity, ktoré pri testovaní je potrebné vykonať, ale nestanovuje časovú súvislosť medzi týmito aktivitami a aktivitami cyklu testovania softvéru. Je to najmä z toho dôvodu, že podľa zástancov X-modelu nemusí každý projekt pozostávať z rovnakých etáp.

Model rozdeľuje testovacie aktivity do dvoch základných kategórií. Prvou je testovanie komponentov, pri ktorom testujeme funkcionalitu základných programových jednotiek a ich spoluprácu na úrovní komunikačných rozhraní. Do tejto kategórie začleňujeme testovanie funkčných blokov. Druhou kategóriou je samotné testovanie systému, pod čím myslíme testovanie integrácie jednotlivých komponentov a ich skupín na základe očakávaného fungovania celého vyvíjaného systému. Z pohľadu všeobecne akceptovaného členenia testovacích úrovní môžeme do tejto kategórie priradiť integračné a akceptačné testovanie.

Model je vertikálne rozdelený na dve časti – ľavá časť je určená na vykonávanie testovania jednotlivých komponentov a pravá na znázornenie aktivít na testovanie celého



Obrázok 3 Zobrazenie X modelu [Zdroj: www.atpjournal.sk]

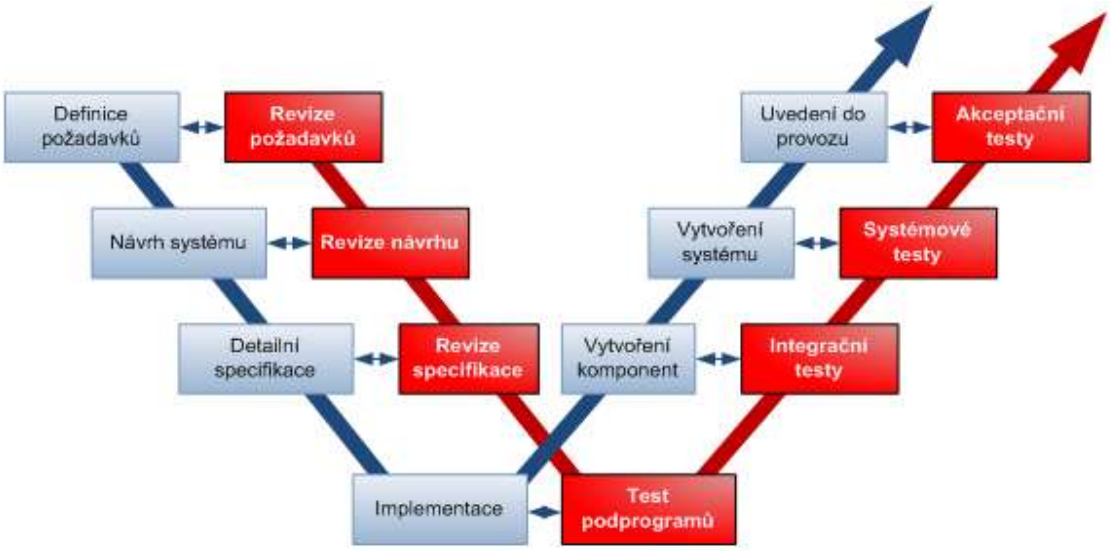
systému. V prípade, že naše vykonanie testov komponentov bolo úspešné, prechádza proces testovania do pravej časti a systém sa testuje ako celok. Aktivity v ľavej časti vykonávame vo firme, kde sa daný produkt vyvíja a aktivity v pravej časti sa môžu vykonávať na tzv. výstupnej kontrole dodávateľskej firmy ale aj priamo u odberateľa softvéru.

1.3.3 W-model

Podkladom pre vytvorenie tohto procesného modelu považujeme V-model, ktorý podľa niektorých autorov V-model nie je dostatočný a to z dvoch dôvodov:

1. Testovacia aktivita na určitej úrovni začína až po začatí vývojovej aktivity na tejto úrovni, prípadne súčasne s ňou
2. V-model sa zaoberá testovacími aktivitami iba na úrovni ich existencie a nedefinuje jednotlivé fázy testovacieho procesu na konkrétnych údajov modelu.

Z horeuvedených dôvodov sa W-model zaoberá plánovaním testovania, jeho vykonaním a následným ladením a procesom odstraňovania chýb, ktorý so samostatným



Obrázok 4 Zobrazenie W modelu [Zdroj: www.atpjournal.sk]

testovaním úzko súvisí. W-model sa skladá z dvoch vetiev na ľavej a z dvoch vetiev na pravej strane modelu. V prvej vetve na ľavej strane znázorňujeme aktivity procesu vývoja softvéru, ale k tejto vetve je priradená ďalšia, ktorá obsahuje tzv. konštruktívne aktivity testovania. Tieto aktivity sa zaoberajú plánovaním testovania, návrhom a tvorbou testovacích scenárov. Pričom podklady pre tieto aktivity sú nielen aktivity na tej istej úrovni, ale aj na tej predchádzajúcej. V pravej časti modelu prvá vetva modelu nám znázorňuje činnosti na vykonávanie testov, tzv. deštruktívne testovacie aktivity a druhá vetva zobrazuje aktivity, ktoré sú zamerané na ladenie testovaného softvéru a odstraňovanie objavených chýb. Tieto činnosti považujeme za reakciu na prípadne chyby zistené pri deštruktívnych testovacích aktivitách a ich funkcionalita je úzko spätá.

Prínosom tohto modelu považujeme fakt, že oddeľuje plánovacie a vykonávacie aktivity testovacieho procesu. Slabšou stránkou tohto modelu je však pravá vetva v pravej časti modelu, ktorá zobrazuje aktivity vykonávajúce ladenie a odstraňovanie chýb. Spôsob, akým aktivity spolupracujú so zbytkom aktivít v modeli nie je dostatočne ilustrovaný. [4]

1.4 Dostupné možnosti vyhodnocovania testovania softvéru

Počas celého procesu testovania softvéru je potrebné kontrolovať a hodnotiť aktuálny stav projektu. Kontrola je dôležitá a jej najbežnejším dôvodom býva aj kontrola kapacít

[4]ATPJournal.sk , 2003. *Procesné modely testovania softvéru*.[Online].[cit. 2018-01-29]. Dostupné na internete: < https://www.atpjournal.sk/buxus/docs/atp-2003-12-17_19.pdf>

testerov a programátorov, ktorí sa podieľajú na testovaní. Pokiaľ sa vyskytuje mnoho chýb, je dobré doplniť tím o ďalších pracovníkov, aby sme mohli dodržať dohodnutý termín odovzdania projektu. V opačnom prípade je možné presunúť voľné kapacity na iné projekty.

Tieto metriky môžeme zobrazovať pomocou grafov pre ich lepšie vyhodnocovanie a zosumarizovanie. Zobrazovať ich môžeme pomocou existujúcich softvérov. V nasledujúcej kapitole vysvetlíme jednotlivé metriky na vyhodnocovanie testov a taktiež aj možnosti použitia existujúcich softvérov, najmä JIRA a HPQC.

1.3.1 Metriky určené na vyhodnotenie stavu testovania softvéru

Poznáme určitý počet druhov metrík, ktoré môžeme použiť na celkové alebo priebežné vyhodnotenie stavu testovania softvéru. Medzi základné metriky kvality softvéru môžeme priradiť nasledujúce:

- Metriky produktu – popisujúce charakteristiky produktu ako je veľkosť, komplexnosť, návrhové vlastnosti, výkonnosť, úroveň kvality a pod.

- Procesné metriky – môžeme ich využívať pri zlepšovaní vývoja softvéru a procesu údržby. Do tejto skupiny môžeme priradiť metriky ako napríklad efektivita odstraňovania defektov počas vývoja, vzor prírastkov defektov počas testovania alebo doba odozvy procesu opráv.

- Metriky projektu – popisujúce charakteristiky projektu a jeho priebehu. Pomocou metrík môžeme kontrolovať potrebný počet vývojárov softvéru, kontrolovať vývoj personálu počas životného cyklu softvéru, cenu, rozvrh a produktivitu projektu.

K priebežnému hodnoteniu testov môžeme využiť aj tri základné metriky, ktoré sú založené na chybách, testoch a kóde.

Metriky kvality produktu môžeme definovať ako orientované metriky na zákazníka.

- **MTTF – meantime to failure (Stredná doba zlyhania**) – táto metrika ktoré sa často používajú v bezpečnostne kritických systémov ako sú systémy riadenia leteckej dopravy, leteckej elektrotechniky a zbraní. Pokiaľ berieme do úvahy cenu, implementácia je veľmi drahá.[5]

[5]MAŘÍK, R. , 2007. *Metriky softwarové kvality*.[Online].[cit. 2018-01-20]. Dostupné na internete: < http://labe.felk.cvut.cz/~marikr/teaching/Y33TSW_10/13.metriky.pdf>

- **Intenzita defektov** - možné definovať ako obecný koncept rýchlosti defektov, ktorý vychádza z počtu defektov vzhľadom k možnosti vzniku chýb v určitom čase. V menovateľovi preto vystupuje veľkosť softvéru a to nasledujúcimi atribútmi:
    - KLOC (thousandlines of code) ... tisíce riadkov kódu
    - SSI (shippedsourceinstructions) ... predané zdrojové inštrukcie
    - CSI (changedsourceinstructions) ... zmenené zdrojové inštrukcie
    - Problém s vlastným počítaním
        - Počítaj iba spustiteľné riadky
        - Počítaj spustiteľné riadky a definície dát
        - Počítaj spustiteľné riadky, definície dát a komentáre
        - Počítaj spustiteľné riadky, definície dát, komentáre a príkazy riadenia dávok
- **Problémy nájdené zákazníkom** – metrika, ktorá nám meria problemy zákazníka pri používaní dodaného produkut.
    - Metrika problémov sa obvykle vyjadruje pomocou problémov, ktoré sa vzťahujú na používateľa a danú dobu, najčastejšie mesiac (PUM = per user month). V tomto prípade by sme mali monitorovať aj celkový počet problémov zákazníka. Metrika je vyjadrená vzťahom:

$$PUM = \frac{\substack{Celkový\ počet\ problémov\ ohlásené\ zákazníkmi \\ za\ danú\ časovú\ jednotku}}{\substack{Celkový\ počet\ licencovaných\ mesiacov\ pre\ daný\ softvér \\ za\ danú\ časovú\ jednotku}} \quad (1)$$

    - Spokojnosť zákazníka meriame často pomocou prieskumu použitím päťbodovej škály
        1. Veľmi spokojný
        2. Spokojný
        3. Neutrálny
        4. Nespokojný
        5. Veľmi nespokojný

Metriky kvality priebehu testovania považujeme za menej formálne definované, zaznamenávajú nám prírastok defektov behom formálneho testovania.

- **Hustota defektov počas formálneho testovania** – vyššia rýchlosť defektov nájdených behom testovania indikujú vyššiu hladinu chýb v samotnom softvéri počas jeho vývoja
- **Vzor prírastkov defektov počas formálneho testovania**–môže indikovať, že testovanie začalo neskoro, testovacia sada nebola dostatočná alebo že testovanie skončilo predčasne
- **Profil odstraňovania defektov podľa fázy** – táto metrika vyžaduje kontrolu defektov vo všetkých fázach vývojového cyklu
- **Efektivita odstraňovania defektov** – túto metriku môžeme definovať nasledujúcim vzťahom. Vyjadrujeme ju v percentách.

$$DRE = \frac{Defekty\ odstránené\ počas\ fázy\ vývoja}{Defekty\ pretrvávajúce\ v\ produktu} * 100\ \%\ (2)$$

Medzi metriky kvality údržby zaraďujeme:

- **Oprava nevyriešených veci** – vyjadruje nároky na pracovnú záťaž, ktorá je viazaná na údržbu softvéru. V podstate ide o počet ohlásených problémov, ktoré zostávajú otvorené na konci každého mesiaca či týždňa.
- **BMI – backlog management index (riadiaci index nevyriešených vecí).** Výsledok interpretujeme v percentách a vypočítame ho pomocou vzorca.

$$BMI = \frac{Počet\ problémov\ vyriešených\ počas\ daného\ mesiaca}{Počet\ prírastkov\ problémov\ behom\ mesiaca} *\ 100\%\ (3)$$

- **Čas odozvy opravy –** obvykle čas odozvy opravy počítame pre všetky problémy ale aj pre problémy, ktoré sme si predtým rozdelili podľa závažnosti. Počítame ju ako strednú dobu života problém od vzniku až po jeho vyriešenie.
- **Percento delikventných opráv** – pokiaľ doba opravy prekročí stanovený časový limit vzhľadom k jeho závažnosti problém klasifikujeme ako delikvent.

$$Percento\ delikventných\ opráv = \frac{\substack{Počet\ opráv,ktoré\ prekročili\ časových\ limit \\ vzhľadom\ k\ ich\ závažnosti}}{\substack{Celkový\ počet\ opráv\ v\ špecifikovanom \\ časovom\ intervale}} * 100\% \quad (4)$$

- **Validné opravy, kvalita opravy –** metriky, ktoré nám umožňuje merať percento zlých opráv z celkového počtu za stanovený časový interval.

Pokiaľ chceme poznať aktuálny stav testov na danom projekte, môžeme vychádzať z aktuálneho počtu nájdených chýb v systéme. Avšak samotný údaj o množstve nájdených chýb nemá pre nás dostatočný význam, pretože projektový manažér len ťažko môže vyvodzovať dôsledky iba z informácií o 150 nájdených defektov v systéme, mesiac pred plánovaným ukončením testovania softvéru. Z toho dôvodu sa u metrík založených na chybách okrem údajov o počte defektov v systéme, využíva aj rozdelenie chýb podľa funkčných oblastí, v ktorých sa vyskytujú, ich závažnosti na priebeh aplikácie a stavu ich opravy. Z týchto informácií môžeme bez problémov vytvoriť sadu užitočných metrík, ktorých porovnanie výstupov a pomery sú dostatočné pre hodnotenie priebehu testovania. [7]

- **Bug fix rate** – pomocou metriky Bug fix rate vyjadrujeme koľko percent z celkového počtu nájdených chýb bolo už opravených. Do čitateľa dosadzujeme počet akýchkoľvek vyriešených chýb, aj tých, ktorých vyriešenie sme zamietli s odôvodnením, že sa jedná o vlastnosť aplikácie.

$$Bug\ fix\ rate = \frac{Množstvo\ opravených\ chýb}{Množstvo\ nájdených\ chýb} * 100\% \quad (5)$$

- **Efektívnosť testu –** určujeme koľko percent chýb bolo nájdených počas testovania softvéru. Tento údaj nám slúži na porovnanie jednotlivých testov alebo na porovnanie s časom stráveným testovaním. Testy, pri ktorých sme zistili, že dlhodobo neprinášajú výsledky, sú nahradené inými, efektívnejšími.

$$Efektívnosť\ testu = \frac{Množstvo\ nádejných\ chýb\ pomocou\ testu}{Množstvo\ celkovo\ nájdených\ chýb} * 100\% \quad (6)$$

- **Rýchlosť objavenia chyby** –výsledok nás informuje o tom, koľko chýb je priemerne nájdených za určitú časovú jednotku. Tento údaj nemôžeme preceňovať. V praxi spravidla táto hodnota je najvyššia na počiatku testovania a postupne klesá s blížiacim koncom testovacej fázy.[11]

$$Rýchlosť\ objavenia\ chyby = \frac{Množstvo\ objavených\ chýb}{Doba\ vykonávania\ testu} * 100\% \quad (7)$$

1.3.2 Porovnanie existujúcich softvérov na vyhodnocovanie testovania SW

1.3.2.1 JIRA

JIRA môžeme definovať ako nástroj navrhnutý špeciálne pre softvérové tímy a to spoločnosťou Atlassian. Podľa oficiálnej stránky spoločnosti sa spája výkonný workflow JIRA s najdôležitejšími prvkami agilného vývoja ako sú flexibilné scrum a kanban boardy, reporty v reálnom čase. Nastavuje nový štandard pre veľký softvérový vývoj. JIRA softvér je tiež prepojený na všetky vývojárske nástroje, ktoré firmy používajú, využíva prebudovaný systém, ktorý prináša informácie o stave vývoja projektu v každej fáze – a to všetko iba na jedno kliknutie.

Je potrebné, aby sme pri opise začali s nástrojmi, ktoré sú dostupné v pôvodnej verzii JIRA prostredníctvom platformy Atlassian Marketplace. JIRA ako reportovací nástroj ponúka nasledujúce prehľady/reporty, ktoré sledujú priebeh projektu:

- **Štandardné prehľady** – najdôležitejšou súčasťou štandardných prehľadov je reportovanie hneď v niekoľkých formách. Reporty opisujúce stav projektu sú nám k dispozícii priamo na úvodnej obrazovke vybraného projektu. Nástroje – grafy a voľne upravovateľné filtre môžeme pridať a usporiadať do panelov a vytvoriť si prehľad o stave projektu, prípadne o stave testovania, ktorý práve potrebujeme. Tieto prehľady nemusíme používať iba na konkrétny projekt, ale aj na svoje vlastné potreby. JIRA nám dáva možnosť si filtrovať a vytvárať si zoznamy úloh, ktoré sú na nás pridelené a je potrebné ich vyriešiť.

- **Projektové prehľady** – keď si v JIRA vytvoríme projekt, automaticky sa nám nastaví niekoľko štandardných dashboardov. Informácie, ktoré poskytujú nám umožňujú

pochopiť štruktúru projektu a jeho jednotlivé fázy na vyššej úrovni. Taktiež opisujú stav práce a aj aktuálne aktivity, prípadne vzniknuté problémy. Tieto prehľady sú určené najmä pre projektový manažment. [12]

1.3.2.2 HP Quality Centre

HP Application Lifecycle Managment (HP ALM) je komerčný softvér pre riadenie životného cyklu aplikácie od firmy Hewlett Packard, jeho súčasťou je HP Quality Center, ktorý je na trhu ponúkaný aj ako SaaS.

Na vyhodnotenie priebehu testovania nám slúži možnosť vygenerovať grafy a reporty, v ktorých sa môžu nachádzať najrôznejšie informácie, ako napríklad koľko testov bolo spustených, koľkokrát boli dané testy spustené a ich výsledky, priebeh a progres testovania v rámci releasu či testovacieho cyklu, a iné. Tieto informácie slúžia test a projektovým manažérom, ktorí ich potom ďalej spracovávajú a interpretujú svojim nadriadeným alebo zákazníkovi, ktorému vyvíjaný systém dodávajú. Vyhodnotenie testov prebieha v module Dashboard (prehľady) a v záložkách Analysis View a Dashboard View.

- **Analysis View** - najprv je potrebné, aby sme vytvorili novú záložku pre graf alebo report, ktorý chceme vytvoriť. V nástroji môžeme nastaviť aj sprístupnenie vytváraných reportov alebo grafov a to uložením buď do zložky Private, kde budú sprístupnené len pre určitých užívateľov, ktorých definuje administrátor alebo do zložky Public, kde budú sprístupnené všetkým užívateľom systému. Na vytvorenie grafu je možné použiť sprievodcu na vytváranie grafov Graph Wizard a vygenerovať rôzne druhy grafov, ktoré môžu byť zamerané výhradne na pokrytie požiadaviek, spustené testy, dosiaľ nespustené testy, zaevidované defekty a iné.

- **Dashboard View** – táto záložka nám slúži na tvorbu súhrnných reportov. Funguje na podobnom princípe ako vytváranie súhrnných reportov v nástroji JIRA. Avšak narozdiel od JIRA nástroja je bežné na vytváranie súhrnných reportov používať iba rôzne typy grafov. [13]

1.3.2.3 Zhrnutie

Vyššie opísané softvérové nástroje sú používateľsky prívetivé na riadenie, testovanie, zadávanie defektov a reportovanie celkového stavu projektu. Aj napriek všetkým ich výhodám je potrebné, aby sme spomenuli, že JIRA a HP Quality Center pracujú, pokiaľ ich majú spoločnosti kúpené v základnej verzii, s aktuálnymi dátami a nie s historickými dátami. Používateľ sa teda dostane k aktuálnym informáciám o stave projektu a testovania, ale nemá celkom jasný prehľad, ako testovanie resp. projekt časom pokročil. Tento nedostatok využívajú test a projektoví manažéri pravidelným exportom dát z JIRA a HP QC do xls súborov a štatistiky udržiavajú mimo softvéru. Problém však môžeme vyriešiť dokúpením a upravením podľa svojich požiadaviek nadstavby systému, do ktorej však spoločnosť musí investovať.

V našej diplomovej práci je cieľom navrhnúť a implementovať doplnok k vyššie opísaným nástrojom na riadenie testovania, ktorý by dokázal nahradiť využívanie excelovských tabuliek test a projektovými manažérmi a držanie dát o stave testovania a projektu mimo softvéru. Doplnok by bol voľne dostupný a v budúcnosti pri jeho realizácií by sa napájal na databázy už existujúcich softvérov. Nami navrhovaný program umožní prístup k základným a súčasne aj detailnejším informáciám o stave projektu a testovania nielen použitím rôznych typov náhľadov, ale aj použitím metrík, ktoré sme opísali bližšie v kapitole *1.4.1 Metriky určené na vyhodnotenie stavu testovania softvéru.* Tieto štatistiky test manažéri budú môcť prezentovať vyššiemu manažmentu a aj samotnému odberateľovi softvéru. Údaje už nebudeme musieť držať mimo softvéru, čím môžeme predísť aj plytvaniu kapacít na ich pravidelnú aktualizáciu. Navrhovaný systém nám pomocou grafov, ktoré budú obsahovať aj historické dáta, umožní náhľad na minulosť priebehu a progresu testovania.

V nasledujúcej tabuľke uvedieme porovnanie vykazovania medzi opísanými systémami a nami navrhovaným systémom z pohľadu poskytovaných funkcionalít podporujúce vyhodnocovanie testovania softvéru koncovým používateľom.

Názov funkcionality	Vykazovanie v JIRA (základná verzia)	Vykazovanie v HPQC (základná verzia)	Navrhovaný systém
Náhľad stavu testovania (počet defektov, uzavretých testovacích scenárov)	Áno (iba aktuálneho stavu)	Áno (iba aktuálneho stavu)	Áno (aktuálneho stavu aj priebehu testovania)
Základné informácie o projekte uvedené priamo v náhľade	Nie	Nie	Áno
Nastavenie užívateľských rolí	Áno	Áno	Áno
Generovanie reportov do xls a pdf formátov	Áno	Áno	Áno
Podpora vytvorenia test plánu	Čiastočná (možnosť generovania exportu)	Čiastočná (možnosť generovania exportu)	Áno

Tabuľka 2 Porovnanie funkcionalít podporujúcich vyhodnocovanie testovania softvéru [Zdroj: Vlastné spracovanie]

2 Technológie použité na analýzu a implementáciu nástroja na vyhodnocovanie testov

Cieľom našej diplomovej práce je navrhnúť a implementovať nástroj, ktorý bude umožňovať vyhodnocovať testovanie softvéru a koncovému používateľovi ponúkne výsledky priebehu samotného testovacieho cyklu. Na vytvorenie návrhu nástroja sme v tejto práci využili nástroj na modelovanie UML diagramov Enterprise Architect od spoločnosti SPARX SYSTEMS. Na vytvorenie webovej aplikácie sme použili webový aplikačný framework ASP.NET MVC v spolupráci s ďalšími technológiami a jazykmi, ktoré podrobnejšie opíšeme v nasledujúcej kapitole.

2.1. UML

UML môžeme definovať ako všeobecne použiteľný vizuálny modelovací jazyk, ktorý používame na špecifikovanie, vizualizáciu, konštruovanie a dokumentovanie artefaktov softvérového systému. Umožňuje nám vytvárať návrhy, ktoré zobrazujú predstavy štandardným, ľahko pochopiteľným spôsobom a poskytuje možnosti na efektívne zdieľanie a komunikáciu týchto predstáv s koncovým používateľom systému, resp. jeho objednávateľom. V podstate zachytáva rozhodnutia ohľadom vytváraných systémov a takisto aj ich celkové pochopenie.

UML môžeme považovať za relatívne otvorený štandard, kontrolovaný otvoreným konzorciom spoločnosti Object Management Group (OMG), ktoré bolo zostavené za účelom vytvárania štandardov, ktoré podporujú interoperabilitu najmä objektovo orientovaných systémov.[Zdroj]

„Samotný jazyk je definovaný niekoľkými dokumentami, ktoré boli publikované OMG. Jeho základné ciele môžeme definovať podľa OMG nasledovne:

- Poskytnúť užívateľom použiteľný vizuálny modelovací jazyk pre tvorbu a výmenu zmysluplných modelov
- Poskytnúť mechanizmy rozšíriteľnosti a špecializácie pre rozšírenie základných konceptov
- Vytvoriť špecifikáciu nezávislú na konkrétnych programovacích jazykoch a procesoch vývoja analýzy

- Poskytnúť formálny základ pre pochopenie modelovacieho jazyka
- Podporovať rozvoj objektových nástrojov
- Podporovať vývojové koncepty ako sú komponenty (components), spolupráce (collaborations), pracovné rámce (frameworks) a vzory (patterns)
- Zahrnúť tzv. Best practice“[6]

2.2. Enterprise Architect

Nástroj Enteprise Architect, ktorý bol vyvinutý spoločnosťou SparxSystems je nástrojom, ktorý má nasledujúce funkcie:

- Navrhovanie a konštruovanie širokej škály softvérových systémov
- Podpora podnikovej analýzy, modelovania obchodných procesov a požiadaviek na správu systémov
- Modelovanie systémov, systémovej architektúry, návrh komponentov
- Vytváranie špecifických modelovacích jazykov založených na jazyku UML
- Vizualizácia širokej škály systémov, procesov, údajov, činností a štruktúr
- Simulácia behaviorálnych procesov, stavových nástrojov a interakcií
- Spolupráca a zdieľanie informácií a modelov
- Testovanie, kontrola kvality a overovanie komplexných systémov
- Riadenie vývojových úloh

Enterprise Architect poskytuje taktiež podporu pre tímový vývoj ale aj pre jednotlivé role, ako sú napríklad analytik, tester, projektový manažér, kontrola kvality a vývojový tím.[5]

2.3. ASP.NET MVC

Základnou myšlienkou MVC architektúry je oddelenie logiky od výstupu. Rieši teda problém tzv. špagety kódu, kedy máme v jednom súbore (alebo v triede) logické operácie a súčasne renderovanie výstupu. Súbor obsahuje taktiež aj databázové dotazy, logiku (C# kód) a rôzne HTML tagy. Všetko je zamotané do seba ako špagety. Takýto kód sa samozrejme zle udržuje a rozširuje o nové funkcionality.

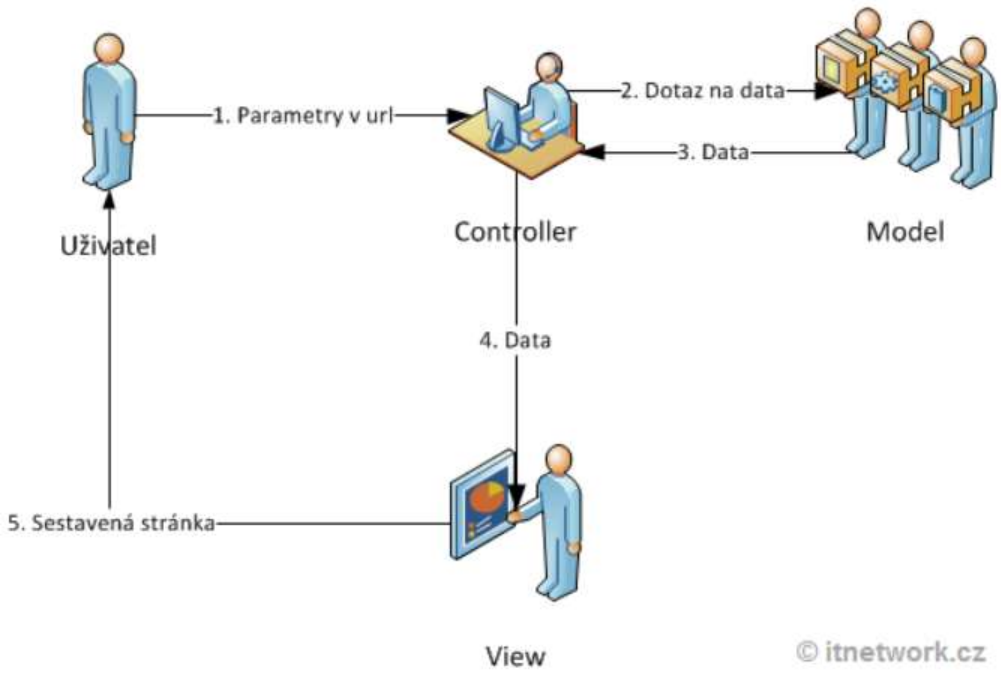
[6] TANUŠKA, Pavel. *Využitie RUP a UML pri tvorbe softvérových projektov.* Trnava: TRIPSOFT, 2007. CD-ROM. ISBN 978-80-89291-10-6 [cit. 2018-03-03]

Celá aplikácia môžeme rozdeliť do troch typov komponentov – Model, View (náhľad) a Controller.

Model obsahuje logiku a všetko, čo do nej spadá. Môže ísť o výpočty, databázové dotazy, validácie a iné. Môžeme povedať, že modely sú zástupcami klasických tried v C#. Jeho funkcia spočína v prijatí parametrov, ktoré sú zadávané koncovým používateľom a v poskytovaní dát. Model však nevie, odkiaľ dané dáta v parametroch pochádzajú a ani ako budú výstupné dáta sformátované a vypísané na stránke. Vďaka EntityFramework sú modely priamo naviazané na tabuľky v databáze.

Pomocou komponentu View zobrazujeme výstupy koncovému užívateľovi. Jedná sa o HTML šablónu, ktorá obsahuje HTML stránku a tagy špeciálneho jazyka, ktorý umožňuje do šablón vkladať premenné, prípadne cykly alebo podmienky. Šablóny môžeme vkladať aj do seba, aby sme sa zbytočne neopakovali a netvorili duplicity – napr. šablóna s layoutom stránky alebo šablóna s menu. View však nie je len šablóna, je to predovšetkým zobrazovač výstupu. Obsahuje teda minimálne množstvo logiky, ktorá je pre výpis nutná. View podobne ako Model nevie odkiaľ dostal dané dáta a stará sa len o ich zobrazenie koncovému používateľovi.

Samotný Controller môžeme považovať za chýbajúci článok celého fungovania systému. Jedná sa o prostredníka, s ktorým komunikuje koncový používateľ, model a view. Drží celý systém pohromade a jednotlivé komponenty, ktoré sme opísali vyššie, prepája. Jeho fungovanie ako súčasti celého cyklu môžeme ukázať na jednoduchom príklade: Koncový používateľ zadá do prehliadača adresu webovej stránky a parametre, pomocou ktorých nám povie, akú podstránku si praje zobraziť. Túto požiadavku ako prvý zachytí tzv. router. Ten podľa parametrov zavolá controller. Controller podľa parametrov spozná, čo má zobraziť koncovému používateľovi a zavolá model, ktorý užívateľa nájde v databáze a vráti jeho údaje. Taktiež zavolá aj metódu modelu, ktorá napr. vypočíta vek užívateľa. Tieto údaje si controller ukladá do premenných a nakoniec vyrenderuje View. View príjme dáta od controlleru a vloží ich do pripravenej šablóny. Hotová stránka je potom zobrazená koncovému používateľovi. [14]



Obrázok 5 Diagram životného cyklu webstránky [Zdroj: www.itnetwork.cz]

3 Cieľ práce. Metodika práce a metódy skúmania

Hlavným cieľom našej diplomovej práce je objektovo orientovaný návrh a implementácia nástroja, ktorý umožní vyhodnocovať testovanie softvéru a ponúkne nám výsledky priebehu samotného testovacieho cyklu. Nástroj by podľa nášho názoru v budúcnosti dokázal nahradiť časté a komplikované využívanie excelovských tabuliek test a projektovými manažérmi, pričom budeme môcť predísť plytvaniu kapacít na pravidelnú aktualizáciu dát medzi neprepojenými systémami, ktoré sa v dnešnej dobe využívajú.

Súčasťou našej diplomovej práce je aj objektovo orientovaný návrh, ktorý môžeme definovať ako komplexný prístup k tvorbe systémov, medzi ktoré zahŕňame odporúčané postupy pre každý aspekt tvorby aplikácií, ktoré majú spĺňať požiadavky klienta a súčasne majú byť ľahko udržiavateľné a rozšíriteľné. Vybraté body nami koncipovaného návrhu sme implementovali.

Návrh informačného systému v našej diplomovej práci pozostáva z nasledujúcich modelov:

- Návrh funkčných a nefunkčných požiadaviek systému
- Model služieb so scenármi
- Model tried
- Model správania prvkov systému

Z hlavného cieľa našej práce sa odvíjajú aj ďalšie čiastkové ciele, medzi ktoré môžeme zaradiť aj demonštráciu využitia modernej webovej .NET MVC platformy, v ktorej sme aplikáciu implementovali. Taktiež naším cieľom bolo poukázať na možnosť využitia iných súčasných programovacích prostriedkov používaných v praxi.

Informácie sme čerpali z rôznych dostupných knižných a internetových zdrojov. Hlavným zdrojom bola dostupná literatúra týkajúca sa modelovania informačného systému. Analýzou a porovnávaním dostupných nástrojov na vyhodnocovanie testov sme získali prehľad o funkciách, ktoré tieto systémy ponúkajú. Informácie sú uvedené na webových stránkach spoločností, ktoré dané softvéry vyvíjajú.

Pri spracovaní našej diplomovej práce sme využili metódy skúmania ako je analýza, syntéza, komparácia a implementácia. V prvej časti práce sme sa zamerali na definíciu samotného testovania a priebehu testovacieho cyklu. Potom sme analyzovali dva najpoužívanejšie nástroje na riadenie testovania a možnosti využívania z pohľadu vyhodnocovania testovania softvéru. Na základe zistených nedostatkov sme postupnou analýzou a syntézou vytvorili návrh systému na vyhodnocovanie testovania, ktorý sme v konečnej fáze implementovali.

4 Výsledky práce

4.1 Požiadavky na informačný systém

4.1.1 Špecifické požiadavky na systém

Pri návrhu informačného systému je v prvom kroku potrebné definovať požiadavky, ktoré sú na systém kladené a zhodnúť sa na cieľoch, ktoré ma informačný systém dosiahnuť. Jedná sa o požiadavky z pohľadu využitia systému, ktoré označujeme ako funkčné, a o požiadavky, ktoré sa týkajú samotnej kvality a stability systému – tie označujeme pojmom nefunkčné.

4.1.1.1 Funkčné požiadavky

Informačný systém, ktorý navrhujeme bude slúžiť koncovému užívateľovi ako nástroj na generovanie, vytváranie a prehliadanie reportov, ktoré budú zobrazovať aktuálnu situáciu testovania softvéru.

F01 - Systém bude vykonávať autentifikáciu užívateľov pri prihlasovaní do systému, a to vyžiadaním prihlasovacieho mena a hesla. Po zadaní správnych údajov systém umožní používateľovi sa prihlásiť.

F02 - Každému používateľovi systému bude priradená užívateľská rola s rôznymi právomocami.

F03 - Systém umožňuje všeobecný náhľad na neuzavreté testovacie scenáre a nevyriešené defekty, ktoré vznikli počas jednotlivých fáz testovania softvéru.

F04 - Navrhovaný nástroj bude umožňovať zmenu závažnosti testovacieho scenáru alebo defektu užívateľovi, ktorému boli na to udelené práva, zväčša pôjde o Test manažéra alebo Projektového manažéra.

F05 - Po vybratí príslušného projektu bude systém zobrazovať základné informácie o projekte, počte založených defektov, počte vyriešených defektov a percentuálneho priebehu testovania.

F06 - Pomocou nástroja bude možné, aby užívateľ vedel vytvárať nad danými testovacími scenármi a defektmi filtre, teda bude ich možné filtrovať a zoraďovať podľa závažnosti, testovacieho modulu, prípadne testovaného podsystému, dátumu zadania do systému.

F07 - Systém bude umožňovať vytváranie prehľadných grafov, ktoré budú obsahovať aj historické dáta. Užívateľovi bude umožnené náhľad na minulosť priebehu testovania.

F08 - Reporty, ktoré budú zobrazené v aplikácii bude možné generovať aj do xls alebo pdf súborov.

4.1.1.2 Nefunkčné požiadavky

N01 - Informačný systém bude prístupný prostredníctvom už existujúcich testovacích nástrojov.

N02 - Informačný systém bude v slovenskom a v anglickom jazyku

N03 - Navrhovaný systém bude používateľsky prívetivý.

N04 - Systém bude dostupný pre používateľov 24 hodín denne.

N05 - Nástroj bude vytvorený ako webová aplikácia, aby poskytovala paralelný prístup viacerým používateľom v rovnakom čase.

4.2 Návrh informačného systému

Na základe analýzy funkčných a nefunkčných požiadaviek sme vytvorili návrh systému za pomoci jazyka UML a dostupných diagramov. V tejto kapitole sa budeme zaoberať samotným návrhom systému v podobe UML diagramov doplnených o textový popis.

V našej práci navrhujeme nasledujúce UML modely:

- Diagram prípadov použitia
- Diagram tried
- Stavové diagramy
- Sekvenčné diagramy

Na zostrojenie vyššie uvedených diagramov sme použili CASE nástroj Enterprise Architect.

4.2.1 Model prípadov použitia Use Case

Diagram prípadov použitia nám pomáha zachytávať funkcionalitu systému z pohľadu používateľa. Prípady použitia opisujú interakciu používateľa s modelovaným systémom,

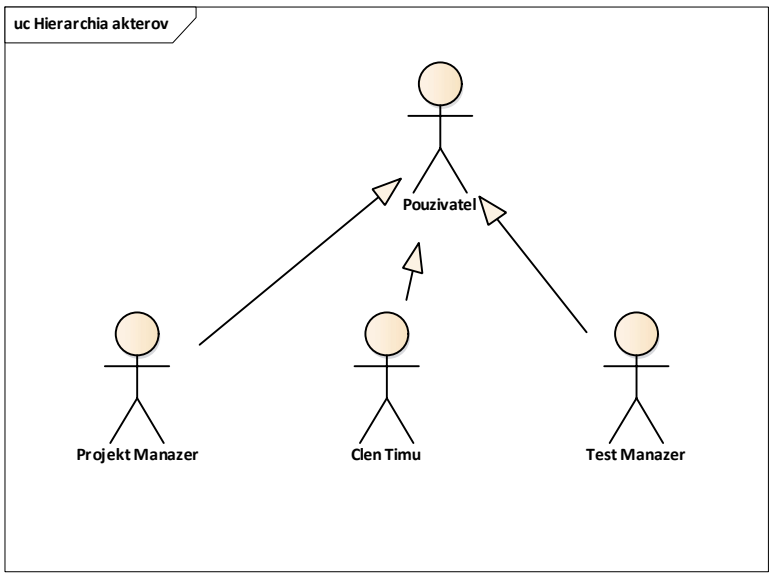
charakterizujú a špecifikujú typickú funkcionalitu daného systému a jeho správanie. Týmto diagramom opisujeme čo systém robí, pričom uvažujeme nad všetkými spôsobmi použitia systému.

Prípady použitia nám reprezentujú typickú množinu scenárov, ktoré niekto nasleduje pri použití systému. Najmä nám pomáhajú porozumieť, štruktúrovať a porovnávať základné požiadavky na informačný systém.

Používateľské role považujeme za dôležitú súčasť systému, ktorá zaisťuje rozdelenie právomocí a možnosť zaobchádzať so systémom na základe typu používateľa. To zabezpečí, že sa k daným informáciám, dátam či funkciám dostane len oprávnená osoba.[4]

Na nasledujúcom obrázku sme zachytili hierarchiu aktérov systému zoradených podľa všeobecnosti od hora nadol, pričom hore je najvšeobecnejšia rola – používateľ. V našom systéme sme definovali nasledujúcich aktérov:

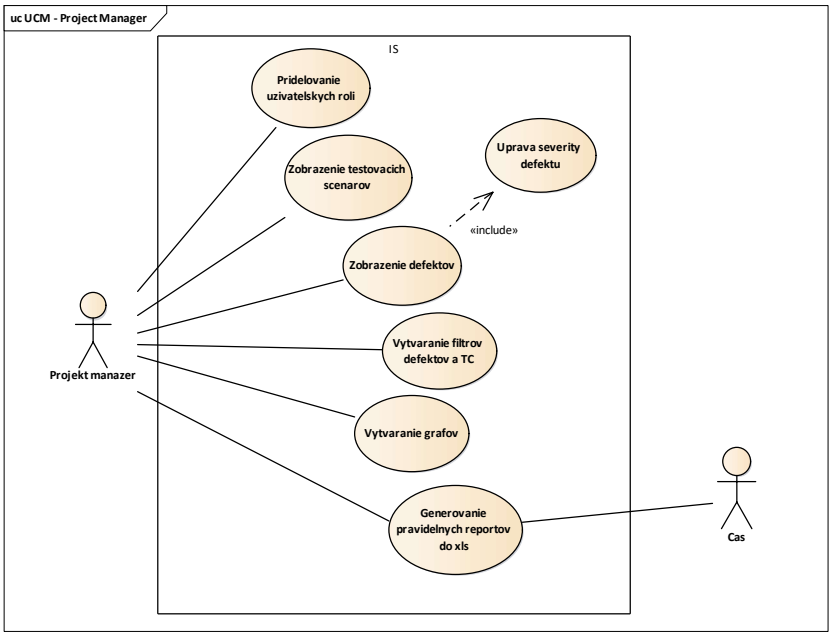
- Projektový manažér
- Test manažér
- Člen tímu



Obrázok 6 Hierarchia aktérov [Zdroj: Vlastné spracovanie]

Projektový manažér

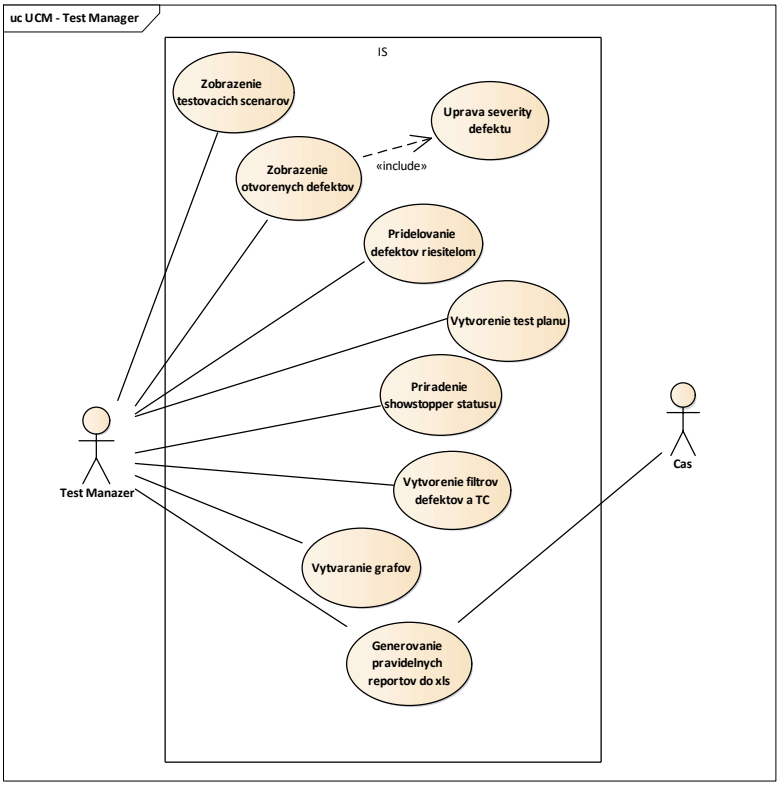
V reálnom svete túto rolu plní vybraný manažér, ktorý je zodpovedný za vedenie projektu vo všetkých jeho fázach. Definuje projektový tím, spracuje plán projektu a potom projekt riadi. V našom systéme projektový manažér prideľuje užívateľské role používateľom, vytvára filtre defektov a testovacích scenárov, grafov a generuje pravidelné reporty do xls súborov.



Obrázok 7 Use Case - Projekt manažér [Zdroj: Vlastné spracovanie]

Test manažér

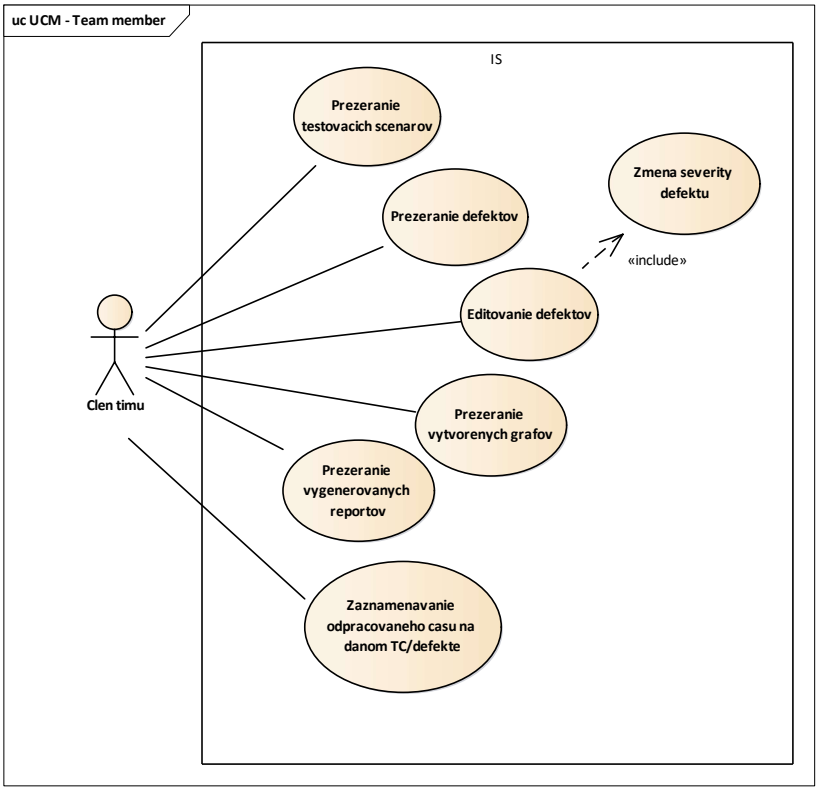
Test manžér v navrhovanom systéme môže navyše od projektového manažéra prideľovať jednotlivé defekty riešiteľom, vytvoriť test plán a priradiť showstopper status k jednotlivým defektom.



Obrázok 8 Use case - Test manažér [Zdroj: Vlastné spracovanie]

Člen tímu

Člen tímu v našom systéme má umožnené prezerať testovacie scenáre, defekty a súčasne ich môže aj editovať. Na rozdiel od manažérov, ktorí riadia testovanie a celkový projekt, nie je mu umožnené vytvárať dashboardy, ktoré zobrazujú aktuálny stav vývoja projektu. V tom prípade mu náš systém umožňuje náhľady na vytvorené grafy a exporty.



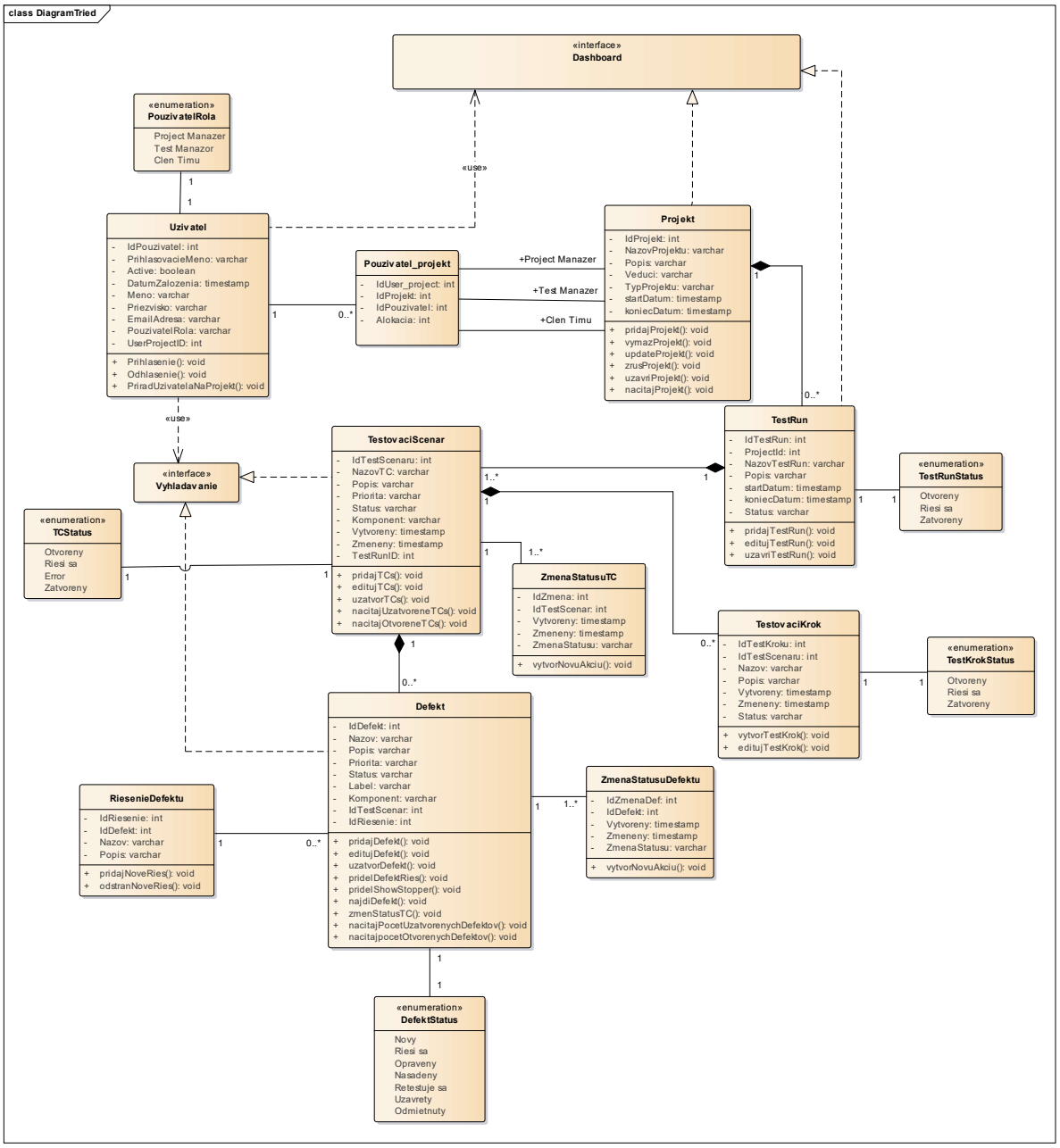
Obrázok 9 Use case - Člen Tímu [Zdroj: Vlastné spracovanie]

4.2.2 Diagram tried

Diagram tried môžeme považovať za základný model objektovo-orientovaného prístupu k analýze a návrhu informačného systému. Ide o model, ktorého hrubá stavba vzniká v etape globálnej analýzy a poskytuje nám základný prehľad o objektoch, ktoré budú v rámci vyvíjaného softvéru vystupovať. Účelom diagramu je zobraziť statický pohľad na vyvíjaný systém, čiže zobraziť vnútorné objekty systému, vlastnosti týchto objektov a vzťahy medzi nimi. Objekt je konkrétny prvok vyskytujúci sa v systéme, ktorý ma definované vlastnosti (pomocou atribútov a hodnôt) a správanie (pomocou operácií, teda postupov určujúcich reakcie na vzniknuté situácie).[4]

Na obrázku môžeme vidieť diagram tried pre navrhovaný informačný systém. V diagrame sú definované nasledujúce triedy: Pouzivatel, Pouzivatel_projekt, Projekt, TestRun, TestovaciScenar, ZmenaStatusuTC, TestovaciKrok, Defekt, ZmenaStatusuDefektu,

RiesenieDefektu. V diagrame sme uviedli aj číselníky: TCStatus, DefektStatus, TestKrokStatus,TestRunStatus a PouzivatelRola. Ide o triedy, ktoré uchovávajú preddefinované číselníky.



Obrázok 10 Diagram tried [Zdroj: Vlastné spracovanie]

Popis jednotlivých tried:

- ***Pouzivatel***

  Medzi objekty tejto triedy zaraďujeme všetkých používateľov, ktorí sú evidovaní v systéme. Primárnym kľúčom je `IdPouzivatel`, ďalšími atribútmi sú systémom vygenerované `PrihlasovacieMeno` a `Heslo`. Takisto obsahuje atribúty ako `Meno`, `Priezvisko`, `EmailAdresa`, `PouzivatelRola` a `DatumZalozenia`.

- ***Pouzivatel_projekt***

  Ide o triedu, ktorá je prepojená s triedami Pouzivatel a triedou Projekt. Je prepojena asociačnými vzťahmi, pričom slúži ako spojovacia tabuľka, ktorá má okrem cudzích kľúčov atribút `Alokacia`. Táto tabuľka v sebe drží informáciu, aké projekty má pridelené používateľ, keďže v našom systéme môže mať používateľ viac projektov, a na projekte môže pracovať viac používateľov.

- ***Projekt***

  Objektami tejto triedy je `NazovProjektu`, `Popis`, `Veduci`, `TypProjektu`, `startDatum`, teda dátum začatia projektu a `koniecDatum`, ktorý nám zaznamenáva informáciu, v ktorý deň projekt skončil. Uvedené metódy zabezpečujú používateľovi, resp. vedúcemu projektu správu projektov v systéme.

- ***TestRun***

  Túto triedu sme spojili kompozíciou s nasledujúcimi triedami a to Projekt a TestovaciScenar, z toho dôvodu, že existencia odkazovaného objektu, čiže v našom prípade TestRun nemá zmysel bez majiteľa, v našom prípade sa jedná o triedu Projekt. TestRun obsahuje atribúty ako `NazovTestRun`, `Popis`, `startDatum`, `koniecDatum` a `Status`, pričom atribút Status môže nadobúdať hodnoty uvedené v číselníku `TestRunStatus`. Pomocou metód `pridajTestRun`, `editujTestRun`, `uzatvorTestRun` je umožnené používateľovi pridať TestRun (testovací cyklus) k projektu, editovať ho alebo ho uzatvoriť po úspešnom/neúspešnom ukončení testovacieho cyklu.

- ***TestovaciScenar***

  Trieda TestovaciScenar by nemohla existovať bez triedy TestRun v systéme a z toho dôvodu sme ju spojili vzťahom kompozícia. Takisto trieda obsahuje atribúty `NazovTC`, `Popis`, `Priorita`, `Status` (atribút môže nadobúdať hodnoty uvedené v číselníku TCStatus), `Komponent`, `Vytvoreny`, `Zmeneny`. Uvedené metódy v diagrame

umožňujú používateľovi editovať testovací scenár ale taktiež aj filtrovať testovacie scenáre.

- ***ZmenaStatusuTC***

  Pomocou triedy ZmenaStatusuTC zaznamenávame zmenu statusu testovacieho scenáru a v ktorý deň daná zmena nastala v našom systéme (v atribútoch `Vytvoreny` a `Zmeneny`)

- ***TestovaciKrok***

  Trieda TestovaciKrok je spojená v diagrame tried kompozíciou s triedou TestovaciScenar, pretože TestovaciKrok nemôže v našom systéme existovať bez prideleného Testovacieho scenáru. Súčasne táto trieda obsahuje aj nasledujúce atribúty: `Nazov`, `Popis`, `Vytvoreny`, `Zmeneny` a `Status` a metódy: `vytvorTestKrok`, `editujTestKrok`, ktoré umožňujú používateľom systému vytvárať a editovať jednotlivé testovacie kroky.

- ***Defekt***

  Defekt trieda je spojená vzťahom kompozícia s triedou TestovaciScenar najmä z toho dôvodu, že Defekt nemôže v našom systéme existovať bez prideleného testovacieho scenáru. Taktiež môže nastať stav, že testovací scenár nemá priradený k sebe žiadny defekt. Defekt pomocou metódy `zmenStatusTC` mení status Testovacieho scenára, `najdiDefekt` metóda nám umožňuje filtráciu existujúcich defektov v rámci projektu. Ako ostatné triedy, trieda Defekt obsahuje atribúty: `Nazov`, `Popis`, `Priorita`, `Status` (môže nadobúdať hodnoty z číselníka DefektStatus), `Label` a `Komponent`.

- ***ZmenaStatusuDefektu***

  Pomocou triedy ZmenaStatusuDefektu zaznamenávame zmenu statusu defektu a v ktorý deň daná zmena nastala v našom systéme (v atribútoch `Vytvoreny` a `Zmeneny`)

- ***RiesenieDefektu***

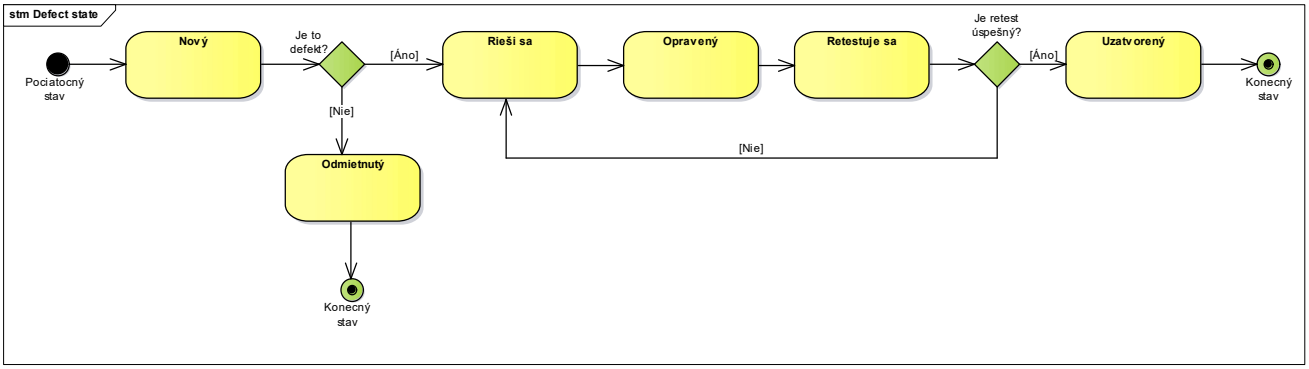
  Trieda v našom diagrame obsahuje nasledujúce atribúty: `Nazov`, `Popis` a metódy: `pridajNoveRies` a `odstranRies`, pričom každý defekt môže mať len jedno riešenie, ale jedno riešenie môže byť pridelené žiadnemu alebo viacerým defektom.

4.2.3 Stavový diagram

Účelom stavového diagramu je zobraziť množinu stavov, ktorými jednotlivé objekty vyvíjaného softvéru prechádzajú v čase a príčiny prechodov medzi stavmi. Objekty v systéme vyvíjajú určité aktivity, teda musia reagovať na prijímané udalosti. Tento diagram je dynamickým modelom vyvíjaného softvéru popisujúcim činnosti jednotlivých objektov. Pomocou stavového je možné získať predstavu o tom, čo objekty jednotlivých tried, ktoré sme zobrazili v diagrame tried, vlastne robia. Získame popis zmien stavov jednotlivých objektov systému a teda aj popis zmien celého systému v čase.[4][5]

V našom systéme mení stav najmä atribút s názvom Status, ktorý môže nadobúdať rôzne hodnoty, ktoré sú bližšie uvedené v jednotlivých číselníkoch a sú popísané v časti *3.2.2 Diagram tried*.

Na obrázku môžeme vidieť ukážku prechodu jednotlivých stavov atribútu Status, ktorý je súčasťou triedy Defekt. Defekt môže nadobúdať stavy Nový, Rieši sa, Opravený, Nasadený, Retestuje sa, Uzavretý a Odmietnutý. Konečný stav defektu je Uzavretý. V prípade, že ho programátor alebo analytik po dôkladnejšej analýze vyhodnotí, že sa nejedná o defekt,



Obrázok 11 Stavový diagram defektu [Zdroj: Vlastné spracovanie]

nadobudne stav Odmietnutý – vtedy ide o jeho konečný stav.

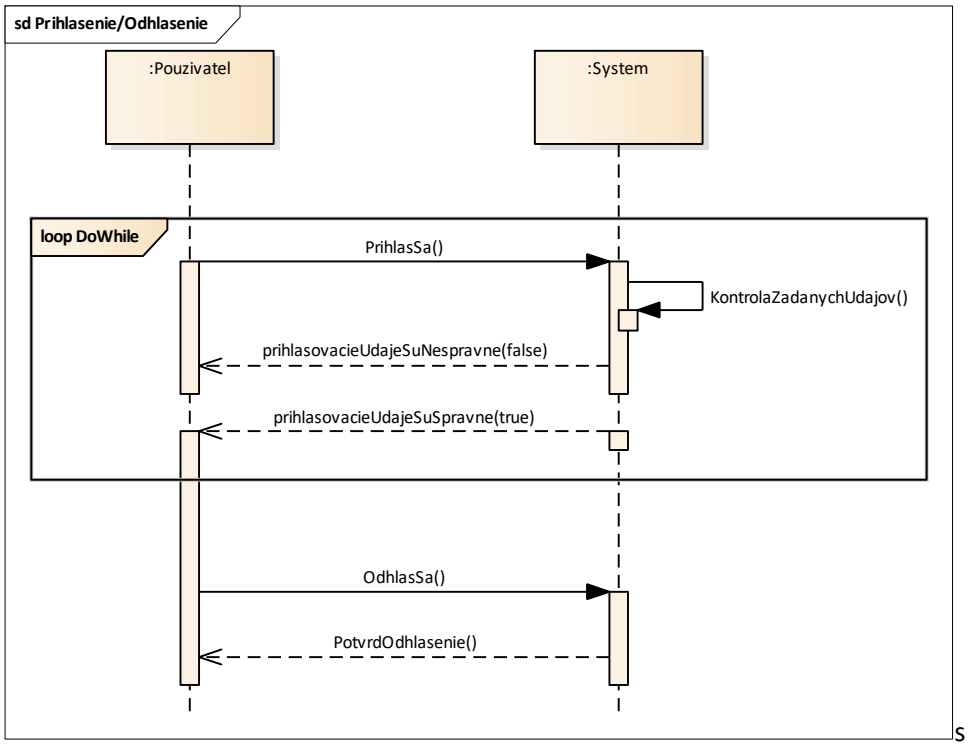
4.2.4 Sekvenčné diagramy

Pomocou sekvenčných diagramov modelujeme dynamické správanie systému, podsystému, triedy a operácie. Diagram znázorňuje tzv. interakcie medzi objektami v časovej

postupnosti, pričom je sémanticky ekvivalentný s diagramami spolupráce, avšak neobsahuje vzťahy medzi objektami. Na nasledujúcich obrázkoch sú znázornené namodelované sekvenčné diagramy služieb systému.

Prihlásenie do systému a odhlásenie

Na nasledujúcom obrázku môžeme vidieť sekvenčný diagram, ktorý znázorňuje prihlásenie a odhlásenie používateľa do systému. V diagrame sú znázornené triedy Pouzivatel a System. Ako prvé zadá používateľ prihlasovacie meno a heslo. Následne systém skontroluje správnosť zadaných údajov. V prípade, že sú údaje nesprávne, cyklus DoWhile sa opakuje a používateľ musí znova zadať prihlasovacie údaje. Ak sú údaje správne, používateľ je prihlásený do systému.

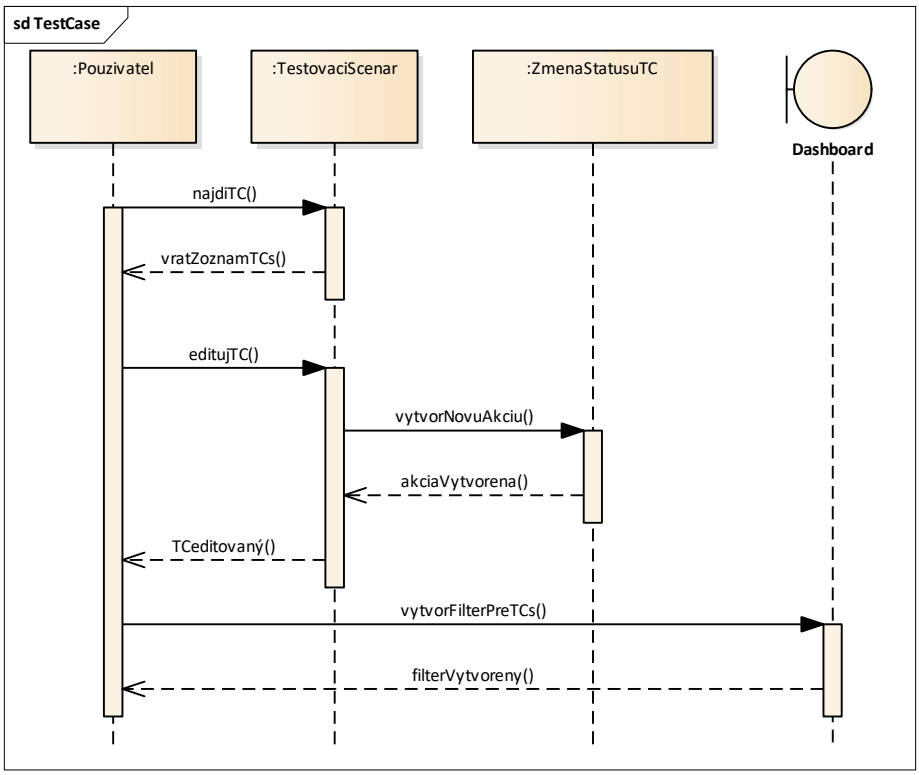


Obrázok 12 Sekvenčný diagram pre Prihlásenie/Odhlásenie [Zdroj: Vlastné spracovanie]

Testovacie scenáre

Na obrázku je zobrazený sekvenčný diagram testovacieho scenára. V diagrame sme uviedli nasledujúce triedy: Pouzivatel, TestovaciScenar a ZmenaStatusuTC. Používateľ požiada objekt TestovaciScenar o vyhľadanie testovacích scenárov a systém mu vráti zoznam

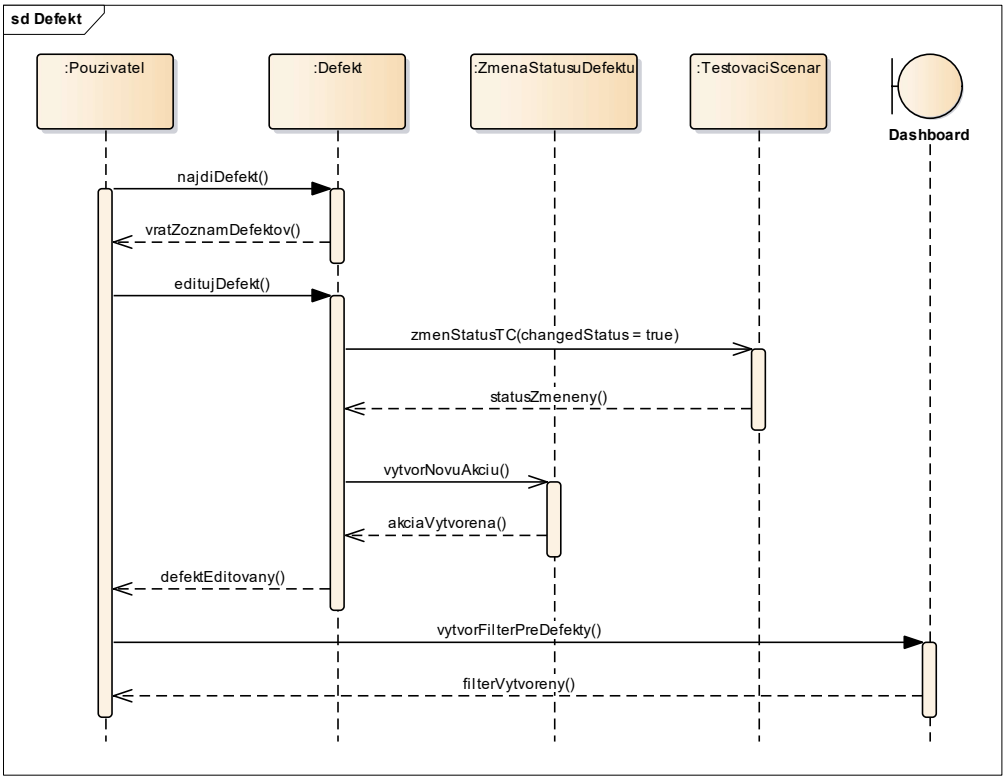
testovacích scenárov. Pokiaľ chce používateľ editovať testovací scenár komunikácia medzi jednotlivými objektami prebehne pomocou metódy editujTC a pri editovaní statusu sa zapíše nová akcia do objektu ZmenaStatusuTC. V prípade, že používateľ chce vytvoriť filter na dashboarde bude zavolaná metóda vytvorFilterPreTCs.



Obrázok 13 Sekvenčný diagram pre Testovací scenár [Zdroj: Vlastné spracovanie]

Defekty

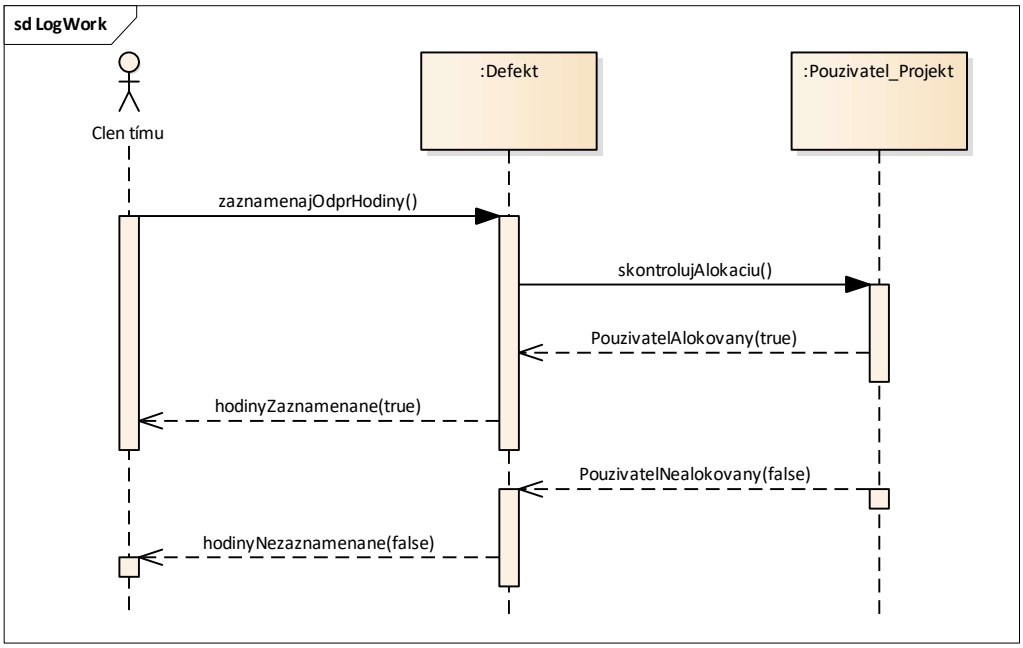
V nasledujúcom sekvenčnom diagrame sme uviedli nasledujúce triedy: Pouzivatel, Defekt, ZmenaStatusuDefektu, TestovaciScenar. Používateľ požiada objekt Defekt o vyhľadanie defektov a systém mu vráti zoznam defektov. V prípade, že chceme defekt editovať, medzi jednotlivými objektami prebehne interakcia pomocou metódy editDefekt. Každú zmenu statusu zaznamenáme pomocou metódy vytvorNovuAkciu. Pokiaľ defekt uzatvoríme, teda pridelíme mu konečný stav, zmení sa stav aj objektu TestovaciScenar a to metódou zmenStatusTC. Pokiaľ chceme vytvoriť filter na dashboarde, zavoláme metódu vytvorFilterPreDefekty.



Obrázok 14 Sekvenčný diagram pre Defekt [Zdroj: Vlastné spracovanie]

Zaevidovanie odpracovaných hodín

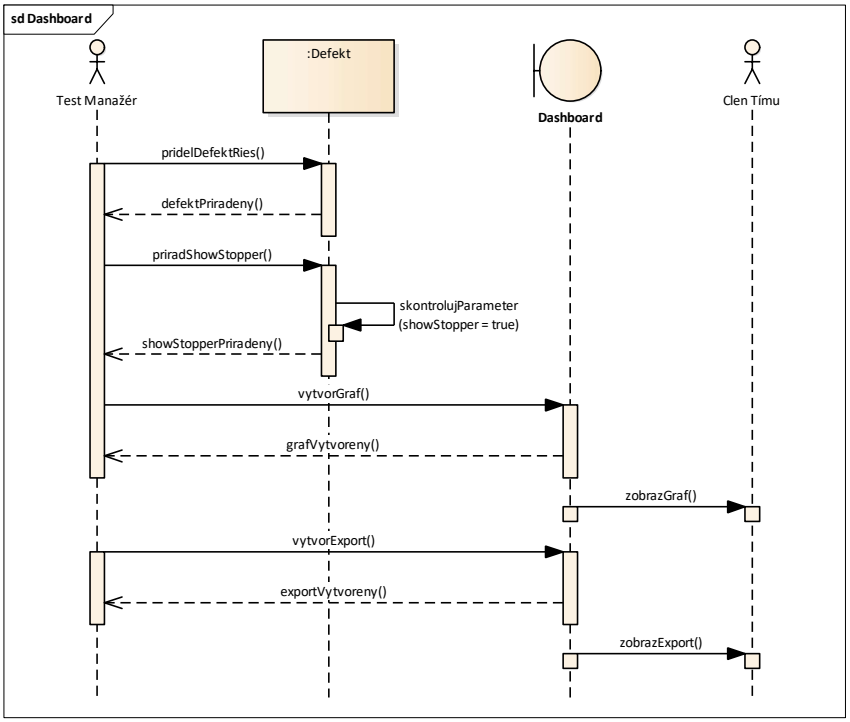
Sekvenčný diagram zobrazený na obrázku nám znázorňuje zaevidovanie odpracovaných hodín na konkrétnom defekte. Aktér Clen timu zaznamená do systému odpracované hodiny na konkrétny defekt. Trieda Defekt požiada triedu Pouzivatel_Projekt, ktorá obsahuje atribút Alokacia o skontrolovanie Alokacie na danom projekte. Pokiaľ je používateľ alokovaný, hodiny budú zapísané. V prípade, že používateľ nie je alokovaný, hodiny nebudú zapísané.



Obrázok 15 Sekvenčný diagram pre Evidenciu odpracovaných hodín[Zdroj: Vlastné spracovanie]

Dashboard

V sekvenčnom diagrame, ktorý môžeme vidieť na obrázku sme znázornili dvoch aktérov systému Test Manazer a Clen Timu. Test manažér pridelí defekt riešiteľovi pomocou metódy pridelDefektRies. Takisto môže prideliť tzv. show stopper defektu. Na dashboarde má možnosť v systéme vytvárať grafy a exporty. Do systému má prístup aj aktér Clen Timu, ktorý si môže grafy a exporty zobraziť.



Obrázok 16 Sekvenčný diagram pre Dashboard [Zdroj: Vlastné spracovanie]

Zobrazenie metrík na dashboarde

4.3 Implementácia informačného systému

Súčasťou tejto kapitoly bude popis implementácie navrhovaného systému – opíšeme proces tvorby databázového modelu od Entitno-relačného diagramu cez Logický model databázy až po Fyzický model databázy, ktorý sme vytvorili v jazyku SQL v programe SQL Server Managemenet Studio. Taktiež sa budeme zaoberať základnými procesmi aplikácie s ukážkami zdrojového kódu. Súčasťou implementácie je aj vytvorenie administrátorskej a používateľskej príručky. Nami vyvinutá aplikácia je dostupná na internetovej stránke: < http://grosschmidtovar-001-site1.gtempurl.com/>. Zdrojový kód k webovej aplikácii je dostupný na adrese: < https://uloz.to/!pLwuopWU090Z/testevaluationtool-rar>

Implementovali sme nasledujúce funkčné požiadavky, ktoré sme definovali v kapitole *4.1.1.1 Funkčné požiadavky.*

F02 - Každému používateľovi systému bude priradená užívateľská rola s rôznymi právomocami.

F03 - Systém umožňuje všeobecný náhľad na testovacie scenáre a defekty, ktoré vznikli počas jednotlivých fáz testovania softvéru.

F05 - Po vybratí príslušného projektu bude systém zobrazovať základné informácie o projekte, počte založených defektov, počte vyriešených defektov a percentuálneho priebehu testovania.

F07 - Systém bude umožňovať vytváranie prehľadných grafov, ktoré budú obsahovať aj historické dáta. Užívateľovi bude umožnené náhľad na minulosť priebehu testovania.

4.3.1 Databázový model

Návrh databázového modelu prebiehal v troch krokoch. Najskôr sme identifikovali jednotlivé entity v systéme a potom zostavili konceptuálny dátový model, ktorý popisuje dáta v databáze bez ohľadu na ich uloženie.

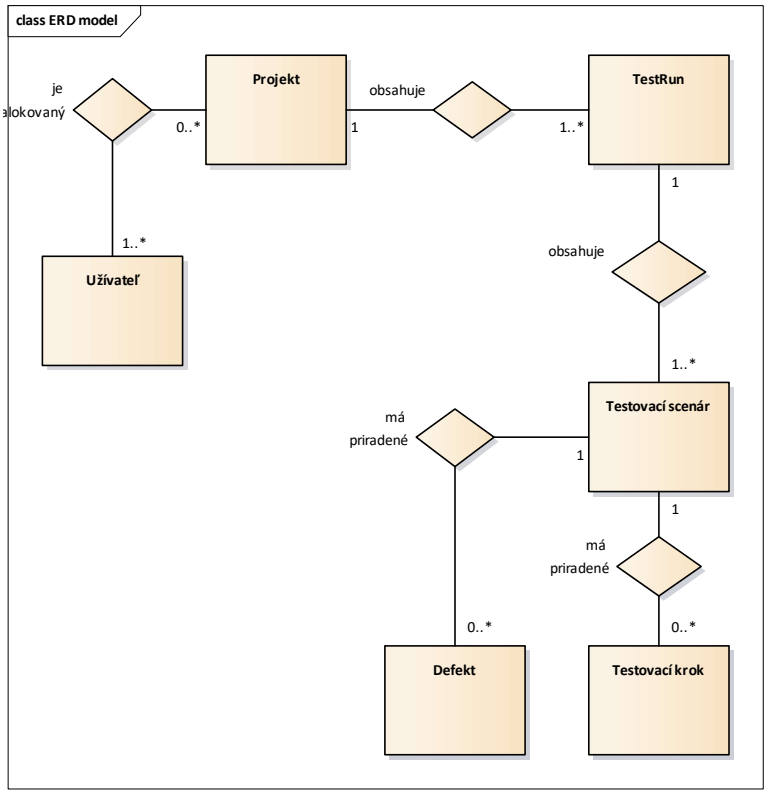
Pri skladbe konceptuálneho dátového modelu sme definovali nasledujúce entity:

- **Užívateľ** – osoba pracujúca v spoločnosti a zaregistrovaná v systéme
- **Projekt** – časovo a vecne ohraničená úloha
- **Test Run** – testovací cyklus
- **Testovací scenár** – súhrn logicky nadviazaných krokov, ktorých účelom je otestovať vybranú funkčnosť aplikácie
- **Defekt** – zaevidovaná chyba softvéru, ktorá je identifikovaná počas testovania
- **Testovací krok** – detailnejšie popísaný krok testovacieho scenáru

Po identifikácii entít sme pokračovali ďalším krokom, ktorým je identifikácia relácii, ktorá existuje medzi jednotlivými entitami:

- **Užívateľ je alokovaný na projekt**. Nie každý užívateľ je alokovaný na projekt a súčasne užívateľ môže byť alokovaný na viac projektov súčasne, preto sme entitu Užívateľ a Projekt spojili vzťahom 1...* ku 0...*.
- **Projekt obsahuje 1 a viac Test run**
- **Test run obsahuje 1 a viac Testovací scenár**
- **Testovací scenár má pridelený defekt.** Nie každý testovací scenár má pridelený defekt a súčasne testovací scenár môže mať pridelených 1 a viac defektov, preto sme entitu Testovací scenár a Defekt spojili vzťahom 1 ku 0...*.

- **Testovací scenár má pridelený Testovací krok.** Nie každý testovací scenár má pridelený Testovací krok a súčasne testovací scenár môže mať pridelených 1 a viac testovacích korkov, preto sme entitu Testovací scenár a Testovací krok spojili vzťahom 1 ku 0...*.



Obrázok 17 ERD model [Zdroj: vlastné spracovanie]

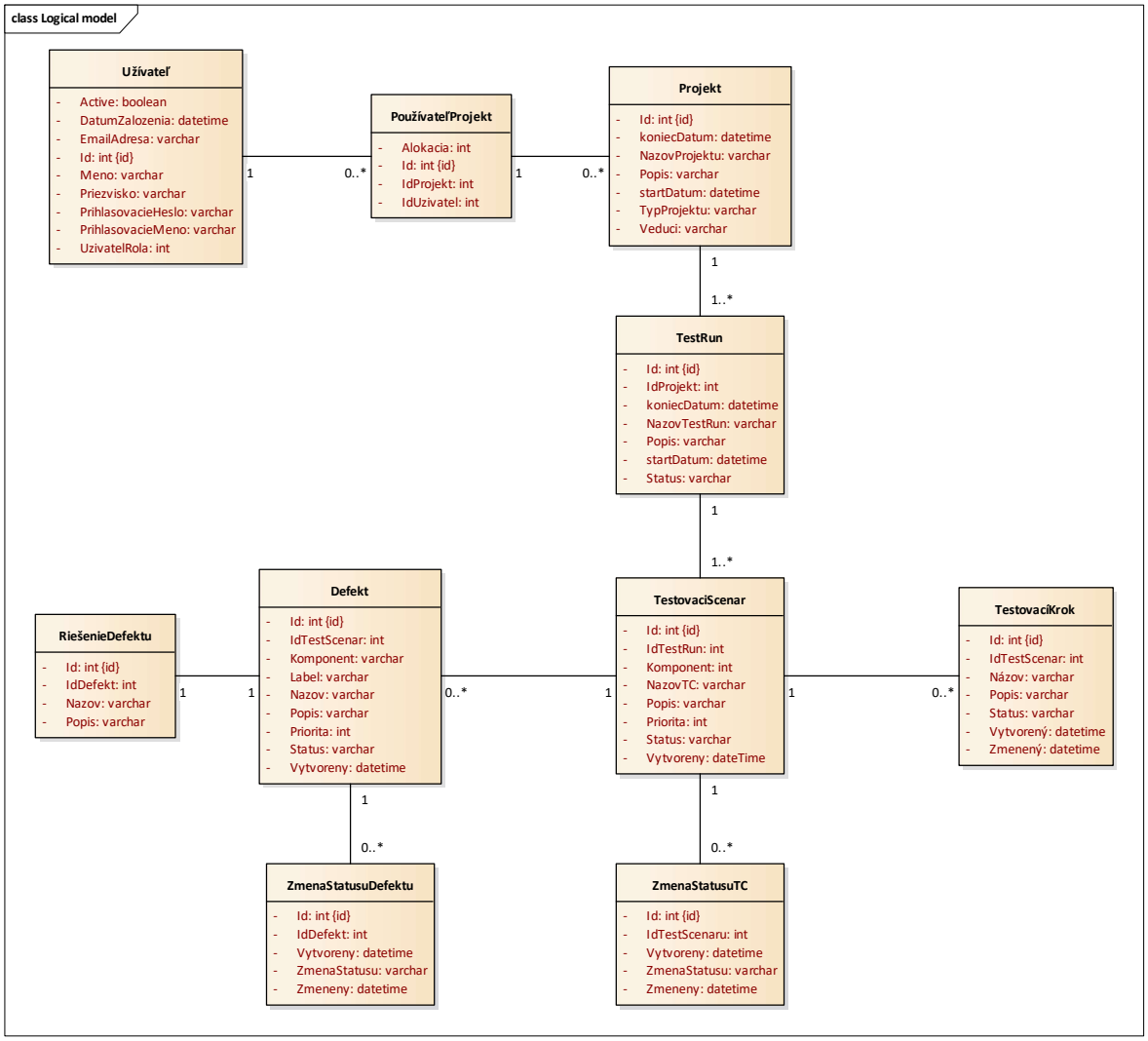
Po návrhu konceptuálneho modelu databázy sme navrhli logický model databázy, pričom sme pri návrhu použili skonštruovaný ER model, ktorý sme si popísali vyššie. Konceptuálny návrh sme previedli do logického návrhu, ktorý je reprezentovaný dátovým relačným modelom. Dáta sme usporiadali do tabuliek pomocou procesu normalizácie. „Účelom normalizácie je kontrola tabuliek vytvorených v predchádzajúcom kroku. Ide o aplikáciu pravidiel použitých v spojení so stĺpcami v tabuľkách. Normalizácia dátového modelu prináša nasledujúce výhody: redukciu redundancie údajov, zvýšenie efektívnosti programovania a vyššiu stabilitu štruktúry údajov.“[7]

Nami vytvorený logický model je v tretej normálnej forme.

[7] : SEMANČÍK, Ľ.: Databázové systémy. 2004. Vojenská akadémia Liptovský Mikuláš, 2004, 115 s. ISBN 80–8040–230–2 [cit. 2018-04-13]

V predchádzajúcom modeli sme mali medzi tabuľkami Užívateľ a Projekt reláciu M:N, a preto sme vytvorili prepojovaciu tabuľku s názvom *PoužívateľProjekt*. Taktiež sme do dátového modelu navrhli nasledujúce tabuľky:

- *ZmenaStatusuTC* – tabuľka obsahuje údaje o každej zmene statusu testovacieho scenáru a kedy bol status zmenený v atribútoch *Vytvoreny* a *Zmeneny*. Je spojená s tabuľkou Testovací Scenár reláciou 1 ku N.
- *ZmenaStatusuDefektu* – tabuľka obsahuje údaje o každej zmene statusu defektu a kedy bol status zmenený v atribútoch *Vytvoreny* a *Zmeneny*. Je spojená s tabuľkou Defekt reláciou 1 ku N.



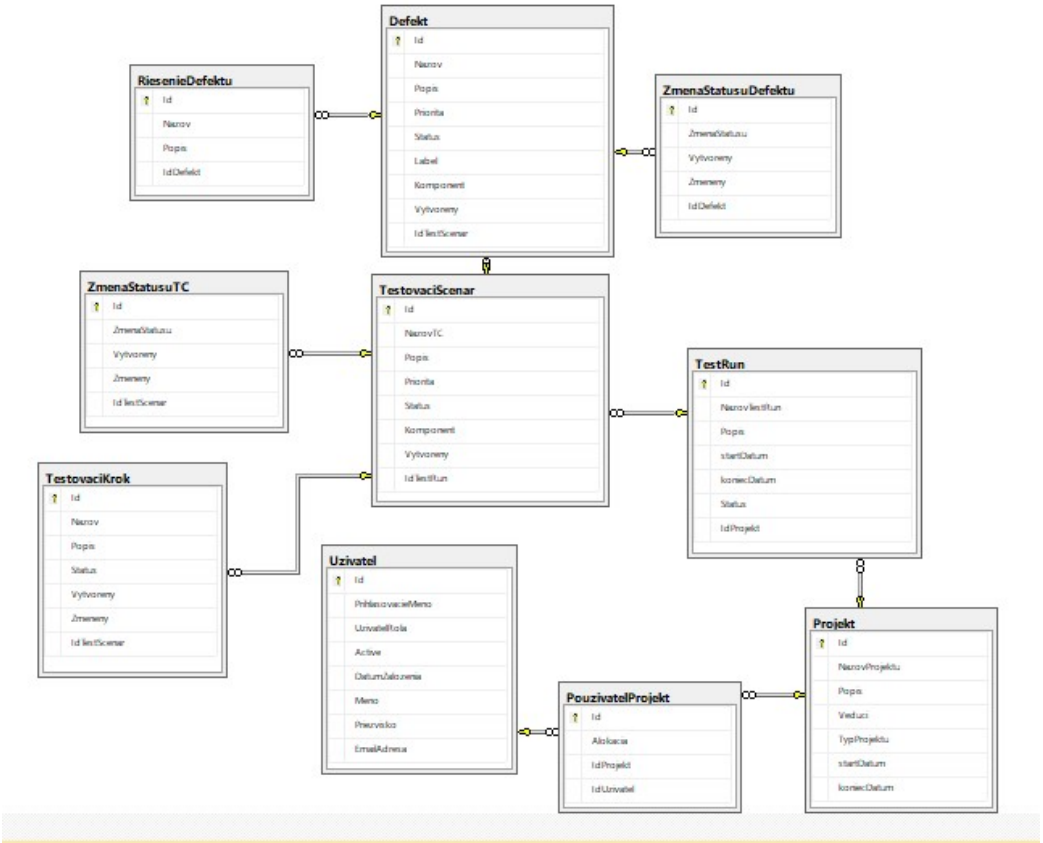
Obrázok 18 Logický dátový model [Zdroj: vlastné spracovanie]

Navrhnutý logický model sme implementovali v cieľovom DBMS (Database management system). Ako cieľový DBMS sme si vybrali program Microsoft SQL Server Management Studio. Jednotlivé tabuľky sme napĺňali dátami a zmeny v databáze boli aktualizované pomocou Script.PostDeployment.sql v Microsoft Visual Studio.

Nižšie uvádzame príklad kódu, ktorý nám slúži na naplnenie tabuľky Uzivatel.

```
if (select count(Id) from dbo.Uzivatel) = 0

begin
   insert into dbo.Uzivatel([PrihlasovacieMeno], [UzivatelRola], [Active],
[DatumZalozenia], [Meno], [Priezvisko], [EmailAdresa]) values('RitaG', 1, 1, '2018-04-
19', 'Rita', 'Grosschmidtova', 'rita.grosschmidtova@gmail.com')
   insert into dbo.Uzivatel([PrihlasovacieMeno], [UzivatelRola], [Active],
[DatumZalozenia], [Meno], [Priezvisko], [EmailAdresa]) values('JanaM', 2, 1, '2018-04-
19', 'Jana', 'Mala', 'malajana@gmail.com')
   insert into dbo.Uzivatel([PrihlasovacieMeno], [UzivatelRola], [Active],
[DatumZalozenia], [Meno], [Priezvisko], [EmailAdresa]) values('IvanG', 3, 1, '2018-04-
19', 'Ivan', 'Gulaty', 'gulaty23@gmail.com')
end
```



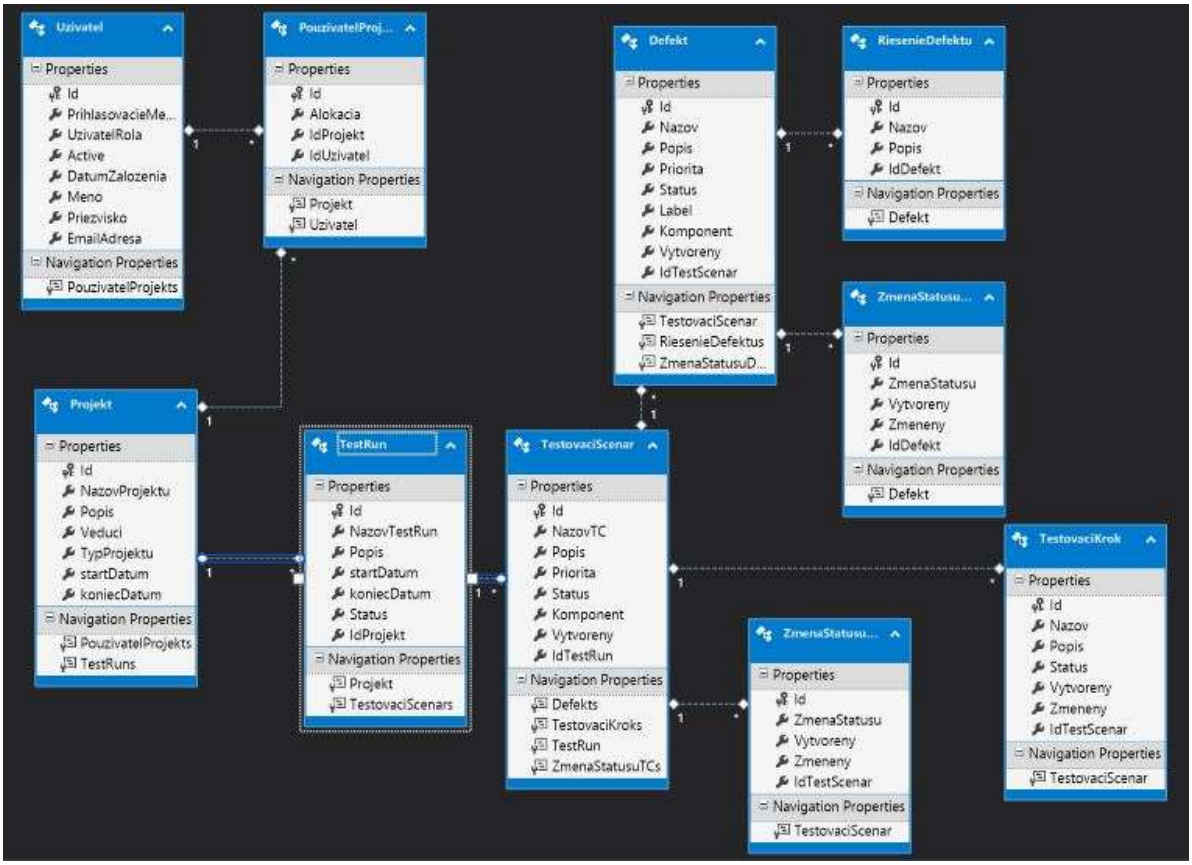
Obrázok 19 Fyzický model databázy [Zdroj: Vlastné spracovanie]

4.3.2 Opis základných procesov webovej aplikácie

Užívateľské rozhranie našej webovej aplikácie sme vytvorili prostredníctvom ASP.NET MVC 6 frameworku. Projekt sa skladá z nasledujúcich troch častí:

- TestEvaluationTool.Dal
- TestEvaluationTool.Db
- TestEvaluationTool.Web

Na prácu s databázou sme použili entity framework, ktorý je súčasťou projektu TestEvaluationTool.Dal, pomocou ktorého sme vygenerovali databázový model a používali ho na prácu s dátami. Navyše nám automaticky vytvoril z databázy triedy (Obrázok 20), ktoré sme využili pri implementácii našej webovej aplikácie. V súbore App.Config sa nachádza na napojenie na databázový projekt TestEvaluationTool.Db.



Obrázok 20 Diagram tried vygenerovaný entity frameworkom [Zdroj: Vlastné spracovanie]

HomeController

V našom projekte pracujeme s ovládačom HomeController, ktorý slúži na spracovanie dvoch navrhnutých dashboardov pre zobrazenie stavu Testovacích scenárov a Defektov na vybranom projekte.

Model

V projekte sa nachádzajú dva typy modelov, ktoré slúžia na uchovávanie a aktualizáciu stavu našej webovej aplikácie, poskytujú nám prístup k dátam, ďalej umožňujú ich ukladanie a aktualizáciu v databáze. Každý model nám poskytuje prístup k relevantným dátam, ktoré sú uložené v databáze.

- DefectsModel
- HomeModel

Testovacie scenáre

Vo funkcii Index, ktorú sme zadeklarovali ako public funkciu s návratovým typom ActionResult sme vytvorili dva objekty projekt typu ProjektTrieda a testScenar typu TestovacieScenareTrieda. Tieto triedy sú implementované v Business logike v časti projektu s názvom Infrastracture.

Zavolaním metódy nacitajProjekt získame detail projektu z databázy. Implementácia uvedenej metódy je uvedená nižšie.

```
public Projekt nacitajProjekt()
        {
            using (var context = new TestEvaluationToolEntities())
            {
                return context.Projekts.FirstOrDefault();
            }
        }
```

Na získanie stavu daného projektu nám slúži vytvorená metóda ziskajStavProjektu. Táto metóda najskôr načíta všetky testovacie scenáre a následne spočíta uzavreté scenáre so stavom rovnajúcim sa 4. Metóda nám vráti percentuálny stav projektu, a to podielom uzavretých testovacích scenárov a všetkých scenárov. Funkcionalita je zapísaná v zdrojovom kóde nasledujúco:

```
public decimal ziskajStavProjektu(int projektId)
        {
            using (var context = new TestEvaluationToolEntities())
            {
```

```
                var scenars = context.TestovaciScenars
                                     .Include(r => r.TestRun)
                                     .Where(r => r.TestRun.IdProjekt == projektId)
                                     .ToList();

                var closedScenarios = scenars.Where(r => r.Status == "4").ToList();

                return Math.Round(((decimal)closedScenarios.Count /
(decimal)scenars.Count * 100), 2, MidpointRounding.AwayFromZero);
            }
        }
```

Funkcia NacitajTC, ktorá je implementovaná v triede TestovacieScenareTrieda získa všetky testovacie scenáre pre daný projekt.

```
public List<TestovaciScenar> nacitajTC(int projektId)
        {
            using (var context = new TestEvaluationToolEntities())
            {
                return context.TestovaciScenars
                              .Include(r => r.TestRun)
                              .Where(r => r.TestRun.IdProjekt == projektId)
                              .ToList();
            }
        }
```

Pre načítanie počtu otvorených testovacích scenárov sme použili metódu nacitajOtvoreneTC a to pomocou dole uvedeného skriptu.

```
public List<TestovaciScenar> nacitajOtvoreneTC(int projektId)
        {
            using (var context = new TestEvaluationToolEntities())
            {
                return context.TestovaciScenars
                              .Include(r => r.TestRun)
                              .Where(r => r.TestRun.IdProjekt == projektId && r.Status
!= "4")
                              .ToList();
            }
        }
```

Podobne sme postupovali aj pri vytváraní metódy, ktorá nám vráti počet zavretých testovacích scenárov, ale v tomto prípade, sme zmenili podmienku nasledujúco:

```
        .Where(r => r.TestRun.IdProjekt == projektId && r.Status == "4")
```

Ovládač následne naplní dáta do objektu typu HomeModel. Tento objekt slúži ako model pre View s názvom Index, ktorý dané dáta vykreslí do webového prehliadača koncovému používateľovi.

Defekty

Vo funkcii Defects, ktorú sme zadeklarovali ako public funkciu s návratovým typom ActionResult sme taktiež vytvorili dva objekty projekt typu ProjektTrieda a defekt typu DefektTrieda. Tieto triedy sú implementované v Business logike v časti projektu s názvom Infrastracture.

V triede DefektTrieda sme vytvorili metódy, ktoré slúžia na načítanie počtu uzatvorených defektov, načítanie počtu otvorených defektov a aj počtu všetkých defektov. Ide o metódy nacitajPocetUzatvorenychDefektov, nacitajPocetOtvorenychDefektov a nacitajPocetDefektov. Dole uvedený zdrojový kód slúži na vytvorenie metódy nacitajPocetUzatvorenychDefektov.

```
public int nacitajPocetUaztvorenychDefektov(int projektId)
        {
            using (var context = new TestEvaluationToolEntities())
            {
                return context.Defekts
                              .Where(r => r.Status == "7")
                              .Include(r => r.TestovaciScenar)
                              .Include(r => r.TestovaciScenar.TestRun)
                              .Where(r => r.TestovaciScenar.TestRun.IdProjekt ==
projektId)
                              .Count();
            }
        }
```

Na ukážku sme si zvolili dve metriky, ktoré sme vo funkcii vytvorili ako objekty DefectFixingEfectivity, ktorá popisuje efektivitu odstraňovania defektov (2) a BugFixRate, ktorá podáva informáciu o koľko percent z celkového počtu nájdených chýb bolo už opravených (5).

```
DefectFixingEfectivity = Math.Round(((decimal)uzavreteBugy / (decimal)otvoreneBugy) *
100, 2, MidpointRounding.AwayFromZero),
BugFixRate = Math.Round(((decimal)uzavreteBugy / (decimal)vsetkyDefekty) * 100, 2,
MidpointRounding.AwayFromZero)
```

Potom ovládač naplní dáta do objektu typu DefectsModel, ktorý slúži ako model pre View s názvom Defects. View dáta vyrenderuje do webového prehliadača.

Vykreslenie grafov

Na vytvorenie grafov sme využili voľne dostupnú javascript knižnicu Chartist.js, ktorú sme doplnili do hlavičky View Layout.cshtml pomocou dole uvedeného skriptu:

```
    <link rel="stylesheet" href="~/Content/chartist.min.css">
    <script src="~/Scripts/chartist.min.js"></script>
```

V príslušnom View sme si vytvorili sekciu na grafy a v ovládači HomeController zavoláme pomocou ajax pre testovacie scenáre metódu TestovacieScenareData.

```
                $.ajax({
                    method: "Get",
                    url: "@Url.Action("TestovacieScenareData")",
                    data: { projektId: $("#projectId").val() }
                })
                    .done(function (result) {
                        var labels1 = result.chart1.labels;
                        var series1 = result.chart1.series;

                        new Chartist.Line('.ct-chart1', {
                            labels: labels1,
                            series: [series1]
                            }, {
                                fullWidth: true,
                                width: '600px',
                                height: '300px'
                            });
```

Kolekcia Dictionary obsahuje tzv. key, ktorý na základe dátumu vytvorenia testovacieho scenáru, ktorý získa z atribútu Vytvorený priraďuje jedinečný identifikátor. Ak je dátum vytvorenia testovacieho scenáru jedinečný, to znamená, že sa v kolekcii nenachádza taký testovací scenár s identickým dátumom vytvorenia, priradíme mu 1.

```
var pocty = new Dictionary<string, int>();
            foreach(var scenar in scenare)
            {
                var key = scenar.Vytvoreny.ToShortDateString();
                if (!pocty.ContainsKey(key))
                    pocty.Add(key, 1);
                else
                    pocty[key]++;
            }
```

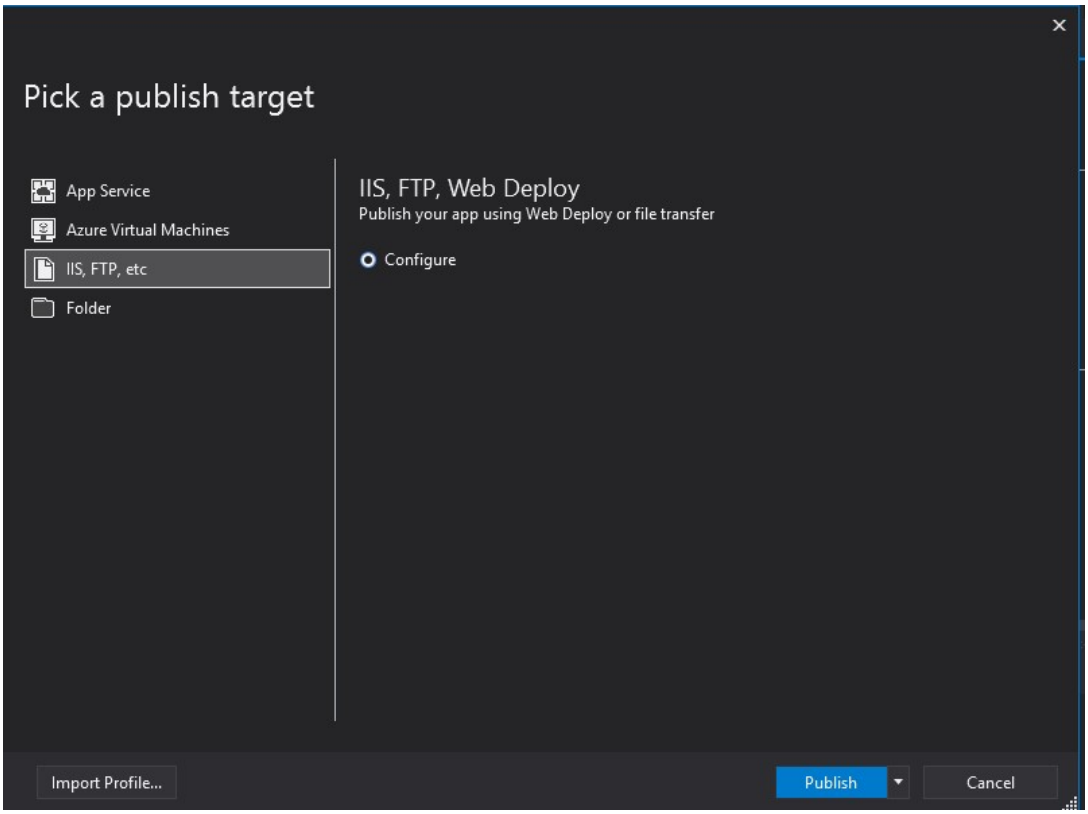
Po skončení cyklu foreach funkcia vráti json so series a labels zabalenými v resulte.

```
            return Json(new
            {
                chart1 = new
                {
                    labels = pocty.Select(r => r.Key).ToList(),
                    series = pocty.Select(r => r.Value).ToList()
                },
```

Nato je vytvorený graf s prislúchajúcimi údajmi vyrenderovaný vo webovom prehliadači. Na rovnakom princípe sme vytvorili aj iné grafy, ktoré sa zobrazujú v aplikácii.

4.3.3 Postup nasadzovania webovej stránky na internet

Pre korektné nasadenie stránky na internet je potrebné, aby sme mali nainštalovaný program Visual Studio (Community Edition verzia 2017). Pravým klikom na projekt TestEvaluationTool.Web sa dostaneme do kontextového menu a klikneme na možnosť PUBLISH. V navigačnom menu sa presunieme na možnosť IIS, FTP, etc a nakonfigurujeme si náš cieľ publikácie (Publish target).



Obrázok 21 Publish target vo Visual Studio [Zdroj: Vlastné spracovania]

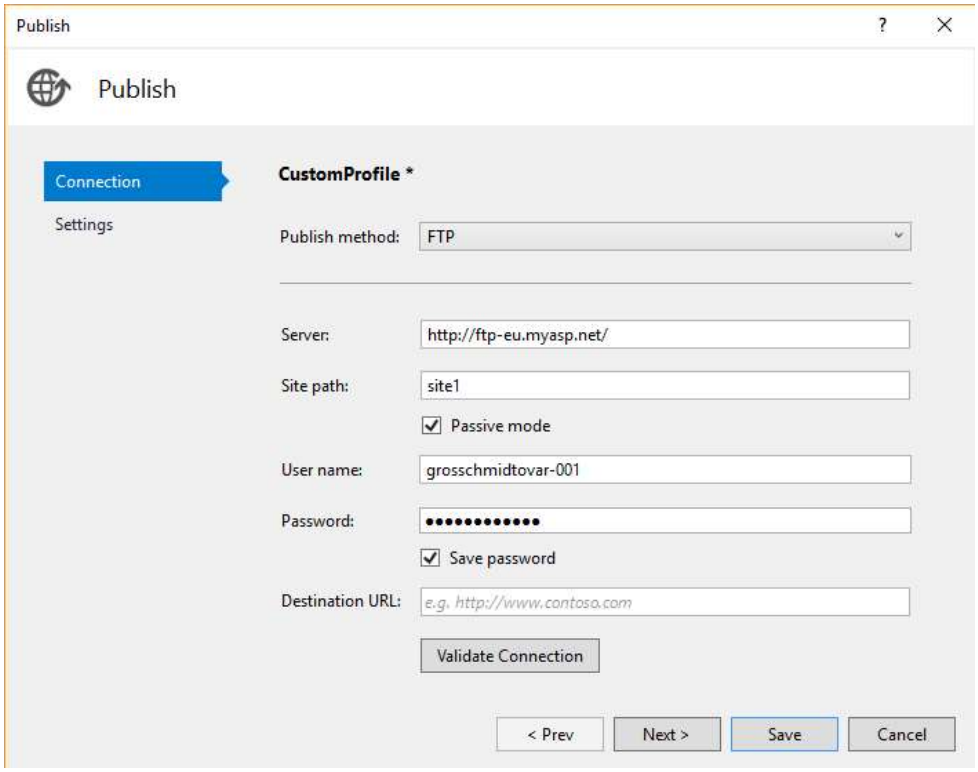
Otvorí sa nám kontextové okno Publish Web, do ktorej zadáme nasledujúce informácie:

Publish method: FTP

Server: http://ftp-eu.myasp.net/

Site path: site1

User name: grosschmidtovar-001



Obrázok 22 Publish Web [Zdroj: Vlastné spracovanie]

A stlačíme tlačidlo Publish. Súčasne musíme na host nasadiť databázu našej webovej aplikácie. Prihlásime sa do vytvoreného konta myasp.net a v databázovom manažéri vytvoríme novú databázu pomocou tlačidla Add database. Automaticky sa nám vytvorí nová databáza, do ktorej je potrebné, aby sme pridali nami vytvorené tabuľky a dáta. Preto v programe Microsoft SQL Management Studio klikneme na existujúcu databázu a vytvoríme Back up, ktorý nám do jedného súboru uloží celú databázu. Vygenerovaný súbor načítame do databázy na myasp.net a cez akciu Restore database databázu nasadíme.

4.3.4 Používateľská príručka

Doplnok na vyhodnocovanie testov bol implementovaný pre fiktívny projekt. Náš projekt má názov Implementácia B2C portálu pre CSOBP a naprogramované náhľady sú určené pre užívateľskú rolu Test manažér.

Pre účely našej diplomovej práce sme databázu naplnili vymyslenými dátami. Statusy sme označili číslami – 1,2,3,4 pre testovacie scenáre a číslami – 1,2,3,4,5,6,7 pre defekty.

Náhľad Testovacie scenáre

Po spustení stránky na hore uvedenej adrese sa ako prvý zobrazí dashboard s názvom Testovacie scenáre. Navigačný panel obsahuje taby, ako sú Testovacie scenáre a Defekty. Vedľa tabov sú zobrazené tri tlačidlá: Upraviť dashboard, Vytvoriť dashboard a Vygenerovať Test Plán. Po kliknutí na tab Defekty sa dostaneme na dashboard, ktorý nám zobrazuje aktuálny stav defektov, ktoré vznikli počas samotného testovania vyvíjaného softvérového produktu.

Na obrazovke máme uvedený názov projektu, vypočítaný percentuálny stav projektu, počet zavretých testovacích scenárov a počet otvorených testovacích scenárov.

Dashboard nám zobrazuje aktuálny stav otestovaných testovacích scenárov v troch grafoch. Prvý graf opisuje počet vytvorených testovacích scenárov v čase. Graf bol umiestnený na našu webovú stránku najmä z toho dôvodu, že na projektoch bývajú aj v rámci jedného testovacieho cyklu importované testovacie scenáre nie naraz, ale postupne. Graf ukazuje krivku koľko scenárov bolo, v ktorý deň založených v testovacom nástroji. Nasledujúci graf opisuje zmenu statusov testovacích scenárov v čase. Teda opisuje priebeh zmeny statusov jednotlivých testovacích scenárov. Posledný graf znázorňuje prehľad aktuálnych statusov testovacích scenárov.

Na spodnej časti stránky sa nachádza prehľadný zoznam všetkých testovacích scenárov, pričom sme zobrazili ich názov, stručný popis, prioritu, status, komponent a kedy bol daný scenár vytvorený v testovacom nástroji [Príloha A]

Náhľad Defekty

Na obrazovku Defekty sa dostaneme kliknutím na tab Defekty v navigačnom paneli. Zobrazí sa nám stránka, ktorá obsahuje názov projektu a stručný prehľad o stave testovania vybraného softvérového produktu. Priebežný stav testovania sme znázornili metrikami Efektivita odstraňovania defektov a Bug fix rate, ktoré nám zobrazujú percentuálnu hodnotu úspešnosti fixovania nájdených defektov testermi. Súčasne sa zobrazuje aj počet otvorených defektov.

Na obrazovke sa ďalej nachádzajú tri grafy, ktoré zobrazujú počet vytvorených defektov v čase, zmenu statusov jednotlivých defektov v čase a prehľad aktuálnych statusov defektov. Rovnako ako pri náhľade Testovacie scenáre aj v tomto náhľade sa na spodnej časti obrazovky nachádza tabuľka všetkých založených defektov na projekte. Tabuľka obsahuje stĺpce názov, popis, priorita, status, label, komponent a vytvorený [Príloha A]

Záver

Hlavným cieľom našej diplomovej práce bol objektovo orientovaný návrh a implementácia nástroja, ktorý bude schopný vyhodnocovať stav testovania počas samotného testovacieho cyklu. Nástroj by mohol nahradiť využívanie excelovských tabuliek manažérmi, ktorí stoja na čele projektu. Mohli by sme tak predísť plytvaniu kapacít na pravidelnú aktualizáciu dát medzi neprepojenými systémami.

V prvej kapitole sme sa zaoberali súčasnou situáciou na Slovensku i v zahraničí v nami vybranej problematike. Na základe nedostatkov porovnávaných nástrojov v teoretickej časti našej práce sme navrhli nový systém, ktorý by predchádzal definovaným chybám. Výstupom diplomovej práce je analýza nového nástroja a taktiež aj implementácia vybratých bodov koncipovaného návrhu systému. Súčasťou výsledkov práce je opis hlavných častí zdrojového kódu a používateľská príručka.

Počas celej doby písania diplomovej práce sme systematicky študovali odbornú literatúru, ktorá bola vybratá na základe nami zvolenej témy práce. Na spracovanie témy v teoretickej a praktickej časti boli použité internetové a knižné zdroje, ktoré sú uvedené v zozname použitej literatúry.

Predpokladáme, že nároky, ktoré sa zvyšujú na spoločnosti a nezastaviteľný technologický pokrok bude korporácie nútiť prejsť na buď platené nástroje na vyhodnocovanie testov alebo sa budú musieť zaoberať vývojom vlastnej aplikácie, ktorá túto problematiku pokryje.

Zoznam použitej literatúry

Bibliografické odkazy na knižné publikácie

[1] SAMUELIS, Ladislav. *Software testing fundamentals.* Košice: Technical University of Košice, 2013. 148s. ISBN 978-80-553-1298

[2] JORGENSEN, Paul C. *Software Testing.* USA: Auerbach Publications, 2013. 494s. ISBN 978-14-6656-0680

[3] TANUŠKA, Pavel. *Využitie RUP a UML pri tvorbe softvérových projektov*. Trnava: TRIPSOFT, 2007. CD-ROM. ISBN 978-80-89291-10-6

[4] NEHÉZ, Matin. *Analýza a návrh informačných systémov.* Košice: Equilibria, 2013. 39s. ISBN 978-80-8143-093-0

[5] MADLEŇÁK, Radovan. *Softvérové inžinierstvo.* Bratislava: Dolis. 2015. 103s. ISBN 978-80-8181-042-8

[6] SEMANČÍK, Ľ.: *Databázové systémy*. Liptovský Mikuláš: Vojenská akadémia Liptovský Mikuláš, 2004, 115 s. ISBN 80–8040–230–2

[7] HUTCHESON, H. *Sofware Testing Fundamentals: Methods and Metrics.* Wiley Pub. 2003. 408s ISBN 047143020X

Bibliografické odkazy na elektronické dokumenty

[7] CASTB – *Učebná osnova pre základný stupeň*, 2013 [Online]. [cit. 2017-12-11] Dostupné na internete:<http://castb.org/wpcontent/uploads/2013/11/ISTQB_CT_Zakladny_stupen_v2011_SK_Beta2.pdf>

[8] Softwaretestinghelp.com,2016. *What is software testing life cycle stlc*.[Online].[cit. 2017-11-30]. Dostupné na internete:< http://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/>

[9] ATPJournal.sk , 2003. *Procesné modely testovania softvéru*.[Online].[cit. 2018-01-29]. Dostupné na internete: < https://www.atpjournal.sk/buxus/docs/atp-2003-12-17_19.pdf>

[10] MAŘÍK, R. , 2007. *Metriky softwarové kvality*.[Online].[cit. 2018-01-20]. Dostupné na internete: < http://labe.felk.cvut.cz/~marikr/teaching/Y33TSW_10/13.metriky.pdf>

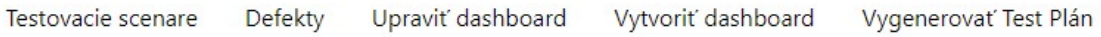
[11] Testovanisoftwaru.cz. , 2010.*Hodnocení testu - metriky*.[Online].[cit. 2018-01-20]. Dostupné na internete: < http://testovanisoftwaru.cz/manualni-testovani/hodnoceni-testu-metriky/>

[12] Confluence.atlassian.com. , 2017.*Reporting in JIRA*.[Online].[cit. 2018-01-02]. Dostupné na internete: < https://confluence.atlassian.com/jirakb/reporting-in-jira-461504615.html>

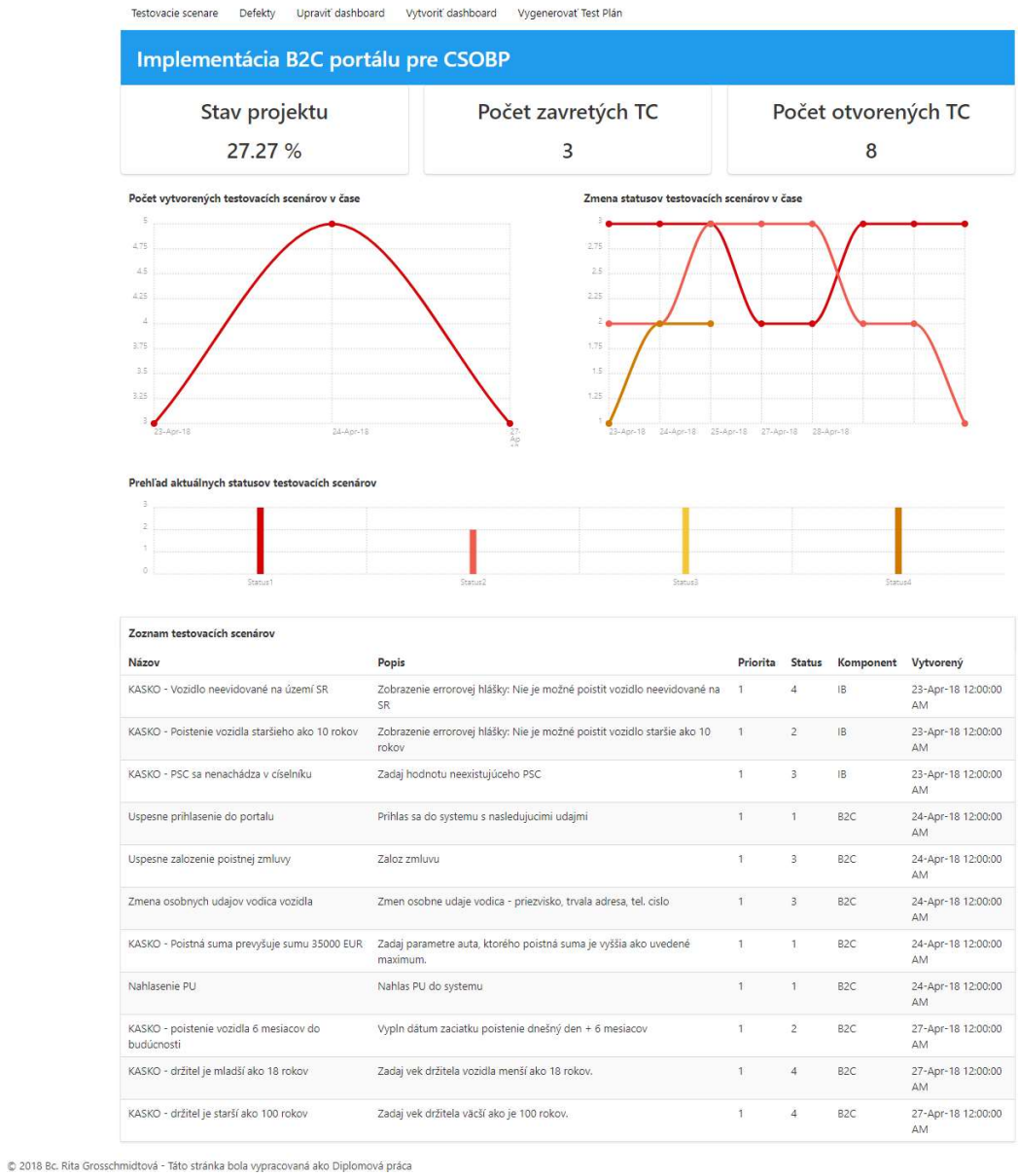
[13] Tutorialspoint.com , 2015.*Reporting in JIRA*.[Online].[cit. 2018-01-29]. Dostupné na internete: < https://www.tutorialspoint.com/qc/qc_dashboard.htm>

[14] Itnetwork.cz. 2017. *ASP.NET Úvod do MVC architektúry*.[Online].[cit 2018-04-24]. Dostupné na internete:<https://www.itnetwork.cz/csharp/asp-net/mvc/asp-dot-net-uvod-do-mvc-architektury >

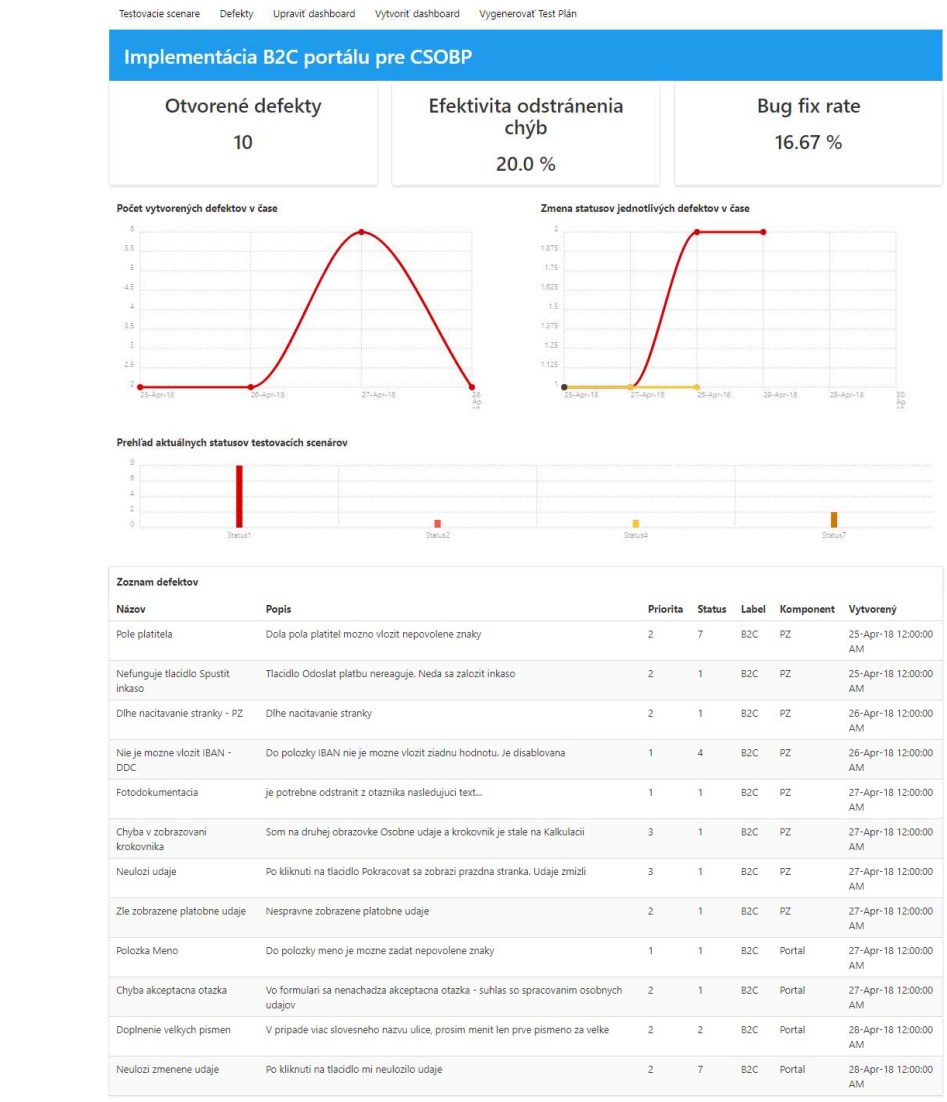
Príloha A



Obrázok 23 Navigačné menu aplikácie [Zdroj: Vlastné spracovanie]



Obrázok 24 Náhľad Testovacie scenáre [Zdroj: Vlastné spracovanie]



Zoznam defektov

Názov	Popis	Priorita	Status	Label	Komponent	Vytvorený
Pole platitela	Dola pola platitel mozno vlozit nepovolene znaky	2	7	B2C	PZ	25-Apr-18 12:00:00 AM
Nefunguje tlacidlo Spustit inkaso	Tlacidlo Odoslat platbu nereaguje. Neda sa zalozit inkaso	2	1	B2C	PZ	25-Apr-18 12:00:00 AM
Dlhe nacitavanie stranky - PZ	Dlhe nacitavanie stranky	2	1	B2C	PZ	26-Apr-18 12:00:00 AM
Nie je mozne vlozit IBAN - DDC	Do polozky IBAN nie je mozne vlozit ziadnu hodnotu. Je disablovana	1	4	B2C	PZ	26-Apr-18 12:00:00 AM
Fotodokumentacia	je potrebne odstranit z otaznika nasledujuci text...	1	1	B2C	PZ	27-Apr-18 12:00:00 AM
Chyba v zobrazovani krokovnika	Som na druhej obrazovke Osobne udaje a krokovnik je stale na Kalkulacii	3	1	B2C	PZ	27-Apr-18 12:00:00 AM
Neulozi udaje	Po kliknuti na tlacidlo Pokracovat sa zobrazi prazdna stranka. Udaje zmizli	3	1	B2C	PZ	27-Apr-18 12:00:00 AM
Zle zobrazene platobne udaje	Nespravne zobrazene platobne udaje	2	1	B2C	PZ	27-Apr-18 12:00:00 AM
Polozka Meno	Do polozky meno je mozne zadat nepovolene znaky	1	1	B2C	Portal	27-Apr-18 12:00:00 AM
Chyba akceptacna otazka	Vo formulari sa nenachadza akceptacna otazka - suhlas so spracovanim osobnych udajov	2	1	B2C	Portal	27-Apr-18 12:00:00 AM
Doplnenie velkych pismen	V pripade viac slovesneho nazvu ulice, prosim menit len prve pismeno za velke	2	2	B2C	Portal	28-Apr-18 12:00:00 AM
Neulozi zmenene udaje	Po kliknuti na tlacidlo mi neulozilo udaje	2	7	B2C	Portal	28-Apr-18 12:00:00 AM

Obrázok 25 Náhľad Defekty [Zdroj: Vlastné spracovanie]