VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

NÁVRH A IMPLEMENTACE QA PROCESŮ UŽIVATELSKÝCH ROZHRANÍ IS
DESIGN AND IMPLEMENTATION OF QA PROCESSES FOR INFORMATION SYSTEM'S UI

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Matej Tábi**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jan Luhan, Ph.D., MSc**
SUPERVISOR

BRNO 2016

Abstrakt
Diplomová práce se zaměřuje na aktuální problémy testování UI v rámci moderních metodik návrhu a implementace informačních systémů. Obsahuje zhodnocení dostupných technologií a postupů na řešení těchto problémů. Zanalyzováno je prostředí vybrané společnosti, procesy Quality Assurance jsou navrženy tak, aby zapadaly do podnikové kultury. Tyto procesy jsou implementovány do konkrétního projektu a také jsou stanoveny metriky, které budou monitorovány a vyhodnocovány v dalším fungování společnosti.

Abstract
The diploma thesis focuses on current problems of UI testing within modern methods of design and implementation of information systems. It includes review of available technologies and pratices to solving these problems. There is analysis of selected company, Quality Assurance processes is designed to fit in the company culture. These processes are implemented into concrete project and also set indicators, which will be monitored and evaluated in future progress of the company.

Klíčová slova
Quality Assurance, testování, javascript, agilné metodiky, podnikové procesy, BPM

Keywords
Quality Assurance, testing, javascript, agile metodologies, corporate processes, BPM

TÁBI, M.*Návrh a implementace QA procesů uživatelských rozhraní IS.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2016. 85 s. Vedoucí diplomové práce Ing. Jan Luhan, Ph.D., MSc.

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 23. května 2016

Bc. Matej Tábi

Obsah

1. Úvod

Pri pojme zabezpečenie kvality sa nebavíme iba o testovaní alebo analýze. Aj keď tento proces dokáže byť náročný, zdĺhavý a nudný, bezpochyby je nevyhnutný. Zabezpečenie, že informačný systém bude pracovať po doručení zákazníkovi, vyžaduje mnoho plánovania a disciplíny. Predvedčiť ho o tom, že aplikácia bude fungovať správne, si však vyžaduje úsilie ešte väčšie.

Čo všetko patrí do zabezpečovania kvality? Samozrejme testovanie je v tomto procese kľúčovou aktivitou. V teoretickej kapitole 3 bude venovaných mnoho strán práve rozboru moderného prístupu k testovaniu. A slovo proces by som chcel zvýrazniť, pretože je jedným z hlavných aspektov QA. Skrz QA je výslednému produktu dodávaná hodnovernosť, resp. zabezpečenie, že produkt bude fungovať správne a ľudia by mali veriť, že bude fungovať správne. Zabezpečenie kvality teda bude pozostávať z krokov, ktoré treba riadiť, aby boli dosiahnuté vytúžené ciele. Charakteristika prístupov riadenia procesov bude preto taktiež súčasťou teoretickej kapitoly 3.

Náplňou nasledujúcej 4. kapitoly bude analýza súčasného stavu fungovania spoločnosti InQool, do ktorej bolo QA nasadzované. Tiež bude uvedený súčasný stav dostupných technológii. V tejto kapitole sa tiež stretnete s analýzou projektu, do ktorého budú procesy implementované. Pri výbere využitých technológii a postupov boli zvažované všetky dôležité pôsobiace faktory.

V praktickej časti budú namodelované konkrétne QA procesy, ktoré sa dočkali aj svojej implementácie. Tiež bude stanovených 8 sledovaných metrík, na základe ktorých môže firma prezentovať kvalitu svojich riešení.

2. Ciele práce a postupy spracovania

Hlavným cieľom tejto diplomovej práce je navrhnúť vhodnú metodiku testovania javaScriptových komponent užívateľského rozhrania s dôrazom na automatizáciu testovania a jeho integráciu do procesov spoločnosti.

Ciele sa dajú rozdeliť do konkrétnych bodov:

1) **Naštudovať problematiku testovania a QA procesov vo všeobecnosti.**

   Štúdium teórie potrebnej k dosiahnutiu hlavného cieľa spočíva v dvoch hlavných celkoch. Prvou časťou teórie je testovanie software, ako a prečo je testovanie pri vývoji potrebné, tiež aký je vzťah medzi testovaním a Quality Assurance. Naštudované boli konkrétne metodiky, ktoré boli využité v praktickej časti. Štúdium testovania bolo zamerané na moderné techniky, agilné metodiky a automatizáciu.

   Druhá časť teórie je zameraná na procesné spracovanie, s tým že na predchádzajúcu časť zameranú na testovanie, plynuje nadviaže. Popísané sú prístupy moderného procesného riadenia, zásady a techniky modelovania a mapovania procesov.

2) **Analyzovať procesy a projetky spoločnosti InQool, zamerať sa pritom na javaScriptové aplikácie.**

   Analýza súčasného stavu spoločnosti spočíva v zdokumentovaní fungujúcich procesov nastavených vo firme. Popísané je doterajšie Quality Assurance v spoločnosti a tiež technológie, ktoré sú využívané na vývoj IT projektov. Pri analýze projektov bol kladený dôraz na JavaScript, pretože užívateľské rozhrania vyvýjaných aplikácii sú vyvíjané predovšetkým v tomto programovacom jazyku. Projekty sú spočiatku analyzované v širšom meradle, neskôr je analýza zameraná na konkrétny jeden pro-

jekt, do ktorého budú neskôr QA procesy implementované.

3) **Analyzovať dostupné technológie a nástroje na testovanie, ako aj na vizuálizáciu a návrh podnikových procesov.**

Analýza dostupných technológii vychádzala z predošlej analýzy. Boli vyberané tak, aby riešenia problému testovania nenarušili už využívané vývojové praktiky. K využívaným technológiám bol uvedený rozbor existujúcich technológii na testovanie v troch úrovniach - jednotkového testovania, integračného testovania a UI testovania.

Uvedený je tiež rozbor dostupných nástrojov na mapovanie a modelovanie podnikových procesov.

4) **Navrhnúť QA procesy a implementovať ich do vývojového cyklu projektu.**

Z prvotného štúdia a vytvorených analýz boli navrhnuté 4 QA procesy. Prvým procesom je spracovanie užívateľských požiadavkov a vytvorenie štrukturovaného zoznamu automatických testov spolu s návrhom testovacích dát a zoznamu manuálnych testov. Druhým procesom je implementácia automatických testov, vymedzené sú zodpovednosti za konkrétne časti. Tretím procesom je manuálne testovanie a vymedzenie prípadov, kedy nebude využívaná pri testovaní automatizácia. Posledným QA procesom je monitorovanie a reportovanie, čiže sa jedná o pasívny proces, ktorý sleduje priebeh vývoja a v prípade chyby je o nej zodpovedný pracovník notifikovaný a podnikne kroky k náprave.

Procesy boli do spoločnosti implementované a ich implementácia je v diplomovej práci popísaná formou prípadových štúdii.

5) **Stanoviť KPI metriky a zabezpečiť ich monitorovanie.**

Po implementácii boli stanovené KPI metriky, ktoré budú sledované a vyhodnocované. Tieto metriky majú slúžiť na vyhodnotenie implementovaných procesov a ako podklad pre ďalšie zlepšovanie a optimalizáciu interných procesov spoločnosti.

3. Procesný prístup k testovaniu SW

Táto kapitola vysvetľuje prečo je testovanie dôležité, ako podporuje vývoj spoľahlivého software a popisuje rôzne typy testov. Druhá polovica je venovaná procesnému riadeniu a celkovo pre túto diplomovú prácu predstavuje teoretické východiská, z ktorých budem vychádzať v ďalšej analýze, návrhu i implementácii.

Dôležitosť testovania

Vývoj software a písanie kódu bez chýb je utópia, chyby a bugy[1] sa vyskytnú a neočakávané situácie nastanú. Postupom času sa požiadavky na software menia a pochybenia sa najčastejšie vynárajú práve počas zmien. V typickom programe sa odhad počtu chýb pohybuje okolo 8 chýb na 100 programových blokov[1]. Niektoré spoločnosti investujú aj polovicu času a zdrojov do ladenia a testovania ich produktov[1].

Testovanie aplikácii má základy v psychologickom vneme dôvery voči produktu alebo aplikácii. Užívateľ môže, pochopiteľne, mať problém ponechať údaje o svojej kreditnej karte spoločnosti, ktorá má problémy s vykresľovaním obrázku, výpisom správy alebo odkazu. V porovnaní webu a natívnej aplikácie, pri práci s webovou aplikáciou je doba pozornosti užívateľa veľmi krátka a očakávaná kvalita je oveľa vyššia v porovnaní s natívnou aplikáciou. V zmysle nestratiť zákazníka prechodom ku konkurencii, je žiadúce zaistiť vysokú kvalitu produktu alebo inak povedané – dostatočne ho otestovať.

Definícia testovania

„Testovanie je proces overujúci program, s cieľom nájdenia čo najvačšieho množstva najzávažnejších chýb.“ [1].

Náklady na opravu chýb počas vývoja nasledujú rovnaké princípy, ako stiahnutie produktu z trhu. Ak sa problémy nájdu v ranej fáze, dá sa zo situácie vyviaznuť pomerne za

[1] softwarová chyba, inak sa jej vraví i defekt, mucha

lacno, zatiaľčo vycúvať z rozbehnutého biznisu býva oveľa nákladnejšie.

Pri testovaní webovej aplikácie existuje niekoľko vrstiev a prístupov, ako systém vyhodnocovať. Najspodnejšiu vrstvu tvorí databáza alebo iné úložisko dát, prostredná vrstva sa označuje ako aplikačná a obsahuje väčšinu logiky aplikácie a najvyššie je práve GUI[2], ktorého úlohou je sprostredkovať kontakt medzi užívateľom a nižšími vrstvami.[2]

3.1. Testovanie GUI informačného systému

Keď je reč o testovaní GUI, myslí sa tým testovanie aplikácie v podobe, s akou bude pracovať užívateľ. Testovanie GUI má mnoho podúloh. Tou najzákladnejšou je samozrejme zistenie, či aplikácia funguje tak, ako má - všetky výpočty, práca s dátami a algoritmy pracujú podľa očakávaní. Táto časť GUI testovania je jednou z mála, ktorú je možné do určitej miery automatizovať. Existuje množstvo nástrojov, ktoré umožňujú simuláciu kliknutia do aplikácie. Týmito testami sa pozornosť sústreďuje skutočne na samotnú podstatu fungovania aplikácie. Preto je použitie podobných nástrojov zmysluplné, sú tu jednoznačne definovateľné vstupy a očakávané výstupy. Treba však brať do úvahy, že prípadný rozdiel reálneho výstupu a očakávaného, môže nastať jak chybou v logike, tak aj chybou zobrazenia informácie na GUI. [5]

Ďalšie testy sa zameriavajú viacej na samotné GUI. Ako bolo spomenuté, GUI sprostredkováva komunikáciu aplikácie s užívateľom – zobrazuje ovládacie prvky, formuláre a výstupy. Testy sa musia zamerať na to, aby boli zobrazené všetky očakávané formuláre, tlačítka, odkazy a výstupy. Testuje sa, či formuláre obsahujú všetky požadované polia, či nad zobrazenými poliami funguje požadovaná validácia (povinné polia, formát atď). U týchto testoch je nutné vziať do úvahy napríklad rôzne správanie aplikácie pre rôzne role užívateľov alebo odlišné zobrazovanie aplikácie v rôznych fázach spracovania (pokiaľ napríklad existuje nejaká forma workflow). [3, 4]

[2]Grafické užívateľské rozhranie

Súčasťou testovania GUI sú i testy, ktoré sa označujú ako testovanie použiteľnosti. Tu dochádza k opusteniu oblasti jasne definovaných požiadavkov a jasne merateľných ukazovateľov. Vedľa samotnej funkčnosti aplikácie je dôležité i to, aby bola pre užívateľa prívetivá, prehľadná – aby ju užívateľ vedel používať. Táto oblasť je pomerne komplikovaná a zložitá. Samozrejme naprogramovať je možné všeličo a tester už zo svojej podstaty dokáže všeličo otestovať. Ale užívateľ, teda zákazník, nemusí byť ochotný výslednú aplikáciu používať. Samozrejme týmto vzniká veľká téma k úvahe – kto a kedy má definovať to, ako má vlastne aplikácia vyzerať a ako sa má ovládať. Je to určite na analytikoch, avšak zákazník nie je vždy schopný sformulovať svoje požiadavky v tejto oblasti. Zatiaľčo požiadavky na funkcionalitu aplikácie sa daria sformulovať pomerne dobre, v oblasti ovládania má zákazník len všeobecnú predstavu a dúfa, že dodávateľ má dostatok skúseností nato, aby niečo vymyslel. Tu pramení mnoho problémov. [3, 4]

Tým sa dostávame k jednému z kľúčových tém testovania GUI. Zákazník totiž ku GUI pristupuje inak ako vývojár. Zatiaľ čo vývojári vnímajú GUI ako niečo, čo prezentuje samotnú prácu a hlavná je logika skrytá pod ním, zákazník má GUI na prvom mieste. Pokiaľ si zázaník organizuje svoje testy aplikácie pred jej akceptáciou, je takmer isté, že prvé hlásené chyby sa budú týkať GUI, a že v celkovom objeme nájdených chýb sa bude väčšina týkať práve GUI. Taktiež hodnotenie závažnosti týchto chýb bude mať zákazník odlišné ako dodávateľ. Dochádza tak k situácii, že GUI chyby budú mať nastavenú najvyššiu prioritu. [4]

Zákazník aplikáciu platí a preto je v poriadku, že ju hodnotí zo svojho pohľadu. Pre vývojový tím je tak nutné si uvedomiť, že otázky okolo GUI nie je vhodné odsúvať a je nutné podobu GUI a jeho riešenie konzultovať so zákazníkom počas celého vývoja. Tým sa dá čiastočne odstrániť možné prekvapenie zákazníka, keď po prvýkrát dostane aplikáciu do ruky. I tak sa nedá vyhnúť tradičným chybám typu požiadavok na inú farbu nadpisov, zväčšenie písma v menu a podobne. [3]

Testovanie GUI má ešte jednu podúlohu, ktorú nie je vhodné zanedbávať. Ide o testovanie kompatibility, čiže o testy, či aplikácia pobeží korektne aj v inom prostredí ako je prednadstavené. U webových aplikácii je dobré otestovať, či funguje správne i na iných prehliadačoch a v rôznch verziách východzieho prehliadača, pre ktorý bola aplikácia vyvinutá. Samozrejme je nutné brať do úvahy požiadavky zákazníka. Ten musí byť schopmý povedať, na akých prostrediach plánuje aplikáciu prevádzkovať – či už konkrétne (obvykle u interných aplikácii, kedy zákazník presne vie aké prostriedky má k dispozícii, prípadne aké vynovenie prostriedkov má v pláne) alebo všeobecne (napr. aplikácia má mať podporu v troch najrozšírenejších aplikáciách v posledných dvoch verziách). Testy kompatibility je treba dobre naplánovať, pretože pri širokej definícii podporovaných prostredí, to môže znamenať podstatný nárast testov. [3]

3.2. Testovanie ako súčasť procesu QA

Vývoj software je proces, ktorý pokiaľ má mať nejaký zmysel, sa riadi určitými pravidlami. Metodík, ktoré sa pri tomto procese môžu uplatniť je celá rada, počínajúc metodikami kladúcimi dôraz na dokumentáciu (RUP) a končiac dnes obľúbenými agilnými metodikami, ktoré sú naopak značne neformálne. Ale nech už je vývoj riadený v duchu ktorejkoľvek metodiky, nezastupitelné miesto v ňom má overovanie kvality vyvíjanej aplikácie.

Voľba vhodnej metodiky vývoja a predovšetkým jej správna implementácia, má zásadný vplyv na kvalitu výsledného software. Testovanie má za úlohu odhaľovať chyby, QA ale slúži k zaisteniu, že výsledný produkt je „kvalitný“. Teda nielen že pracuje tak ako má, neobsahuje chyby, ale napĺňa taktiež ciele, pre ktorý bol jeho vývoj navrhnutý, spĺňa predstavy jeho užívateľov a v neposlednom rade, že jeho vývoj je efektívny.

Zavedenie QA do vývoja software v podstate znamená, že sú jasne definované role a ich zodpovednosti, sú navrhnuté a aplikované metódy verifikácie správnosti „výstupov“ v

jednotlivých fázach vývoja. Teda napríklad spôsob preberania jednotlivých dokumentácii (napr. zákazníckych požiadavkov, analýzy a podobne) alebo spôsobom dodávok hotového kódu a jeho overovaní. Nech už je zvolený prístup formálny a overovanie je v jednotlivých fázach zachycované do príslušných dokumentov alebo naopak neformálny, vždy musí existovať osoba zodpovedná za kvalitu aktuálneho výstupu a taktiež osoba zodpovedná za overovanie tejto kvality.

Malo by byť jasné, že za kvalitu celého produktu nesie zodpovednosť tester. On je tým, kto nakoniec povie, či vytvorená aplikácia je vhodná k používaniu a spĺňa všetko, čo sa od nej čaká. Pokiaľ sa po skončení vývoja v aplikácii objavia chyby, prvé otázky smerujú k testerovi, pretože práve jemu tieto chyby unikli. K tomu, aby si túto zodpovednosť mohol vziať tester na svoje ramená, musí dostať príležitosť odviesť dobre svoju prácu. čiže musí existovať QA začlenené do vývoja, ktoré dá testerovi tú moc kvalitu výsledného produktu ovplyvniť. [7]

Stále bohužiaľ existuje mnoho firiem a vývojových tímov, ktoré si možno dôležitosť testovania uvedomujú, ale nedávajú mu dostatočný priestor pri plánovaní vývoja a tým ani v jeho reálnom priebehu. Veľmi častá je prax, že sa testeri „do hry“ dostávajú až v okamihu, kedy je aplikácia už dokončená, alebo jej vývoj už začal. Sú tak ignorované nebezpečia chýb, vzniknutých z napríklad nesprávnym spracovaním zákazníckych požiadavkov alebo nesprávnou analýzou. Tieto chyby sa potom môžu prejaviť až pri testovaní dokončenej aplikácie a v tej chvíli môže byť ich náprava značným problémom.

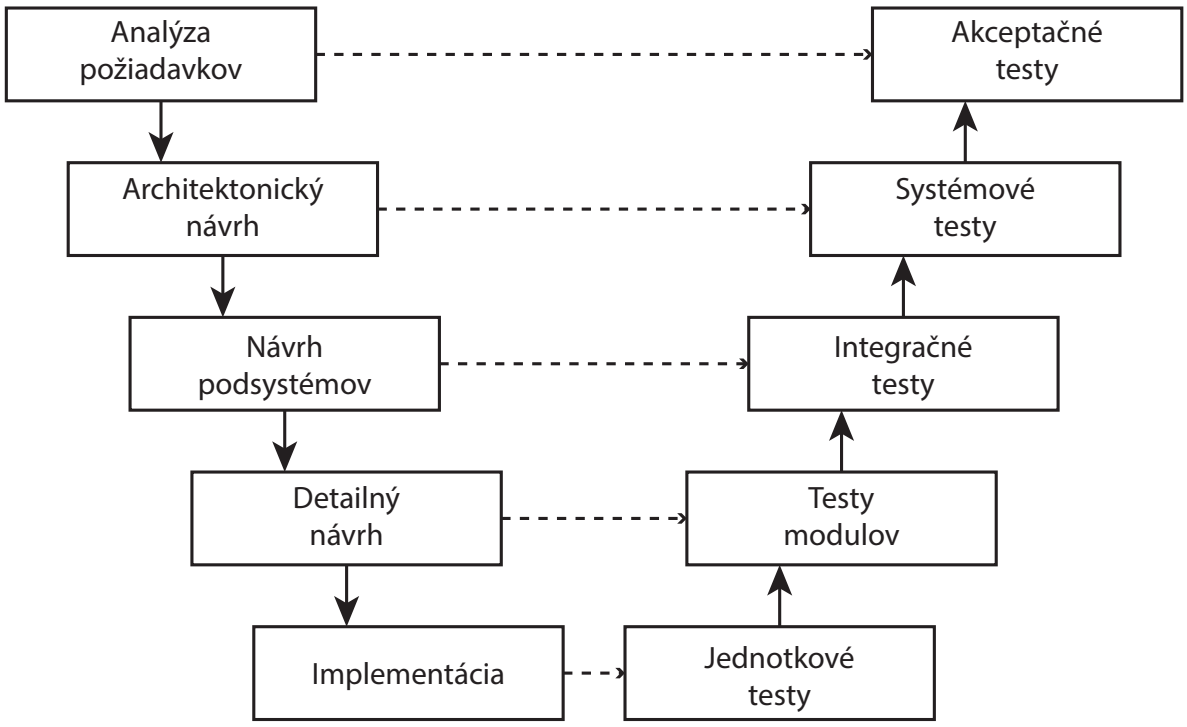
Častou chybou je taktiež podceňovanie časovej dotácie testovania v rámci celého vývoja. Mnoho projektových manažérov berie fázu testovania ako nejaký „buffer“ pre prípad, že by sa ostatné fázy predražili. Uplatňuje sa potom efekt valiaceho sa kameňa, ktorý bohužial príliš často fázu testovania úplne zruší.

Fungujúce QA je to, čomu sa firmy snažia priblížiť. Jedinou cestou ako to dosiahnuť, je aplikovať kontrolu kvality v celom procese vývoja. [6]

3.3. Druhy testovania v procese vývoja SW

Je nutné podotknúť, že testovanie nie je jednorázový proces, ktorý prebehne na konci vývoja software pre uistenie, že je všetko v poriadku. Testovanie je zviazané s celým vývojom a jeho úlohou je dosiahnuť stav, kedy na konci procesu vývoja nebude aplikácia obsahovať žiadne závažné chyby.

Rôzne druhy testovania slúžia v priebehu procesu vývoja k overeniu, že daná fáza vývoja prebehla správne a že výsledky zodpovedajú očakávaniam. Vzťah medzi fázami a druhmi testovania sa často zobrazuje v podobe takzvaného V modelu 3.1. [8]



Obr. 3.1: V-model testovania (vlastné spracovanie podľa [8])

V ľavej časti modelu sú uvedené fázy vývoja aplikácie. V pravej zas jednotlivé druhy testovania. Každý druh testov slúži k overeniu inej fázy vývoja. Základom tohoto prístupu je postup testovania od malých častí aplikácie, cez vačšie funkčné celky, cez integráciu komplexných častí aplikácie, prípadne integráciou viacerých aplikácii až po kompletné otestovanie celej aplikácie. Prechod na ďalší druh testovania predpokladá úspešné dokončenie predchádzajúcej fázy. často sú pre tieto účely v rámci testovania dokumentácie definované konkrétne podmienky, za ktorých je možné príslušný druh testovania spustiť. Táto ilustrácia je však len zjednodušeným modelom, v skutočnosti rozlišujeme viacero druhov testovania.

Jednotkové testy

Ide o prvú fázu testovania, ktorá sa zameriava na najmenšie testovateľné časti aplikácie, známa aj pod pojmom Unit testy. Ide obvykle o testy jednotlivých komponent aplikácie na úrovni modulov, objektov a tried. Tento druh testovania obvykle vykonávajú vývojári a nebýva zahrnutý do plánov testov aplikácie. Vývojári si týmito testami overujú, že nová alebo zmenená časť kódu funguje, resp. nespadne do chyby, a že jej funkcia zodpovedá očakávaniu.

Assembly testy

Tieto testy sú na pomedzí medzi Unit testami a testovaním vykonávaným testermi. Úlohou Assembly testov je overiť, že jednotlivé časti aplikácie je možné „zostaviť“ do funkčného celku. Ide teda o test integrácie jednotlivých komponent. Vykonávať tieto testy môže vývojár a je to celkom bežné. V rozsiahlych projektoch, prípadne pri zložitejšej štruktúre procesov vývoja, býva určený konkrétny pracovník, ktorý je primárne zodpovedný práve za zostavenie aplikácie a Assembly testy.

Systémové a integračné testy

Vo V-modeli je najpodstatnejšia časť testovania zhrnutá práve do tejto oblasti. Toto rozlíšenie je však príliš hrubé a popis tejto skupiny testov bude rozdelený podrobnejšie.

- **Smoke testy**

  Predtým ako je spustené testovanie aplikácie je dobré overiť, či je táto aplikácia vôbec k testovaniu vhodná. Tým je myslené predovšetkým to, že aplikácia je nainštalovaná, spustená, prístupná a nakonfigurovaná pre potreby testov. Ide o rýchle testy, obvykle obsahujúci jednoduchý „priechod“ skrz aplikáciu, ktoré dokážu overiť všetky spomenuté atribúty. Tieto testy sú už v pôsobnosti testerov. Je ale možné, že ich vykonáva pracovník zodpovedný za správu testovacích prostredí.

- **Integračné testy**

  U integračných testoch je nutné rozlišovať medzi integráciou vnútornou, ktorá spočíva vo vzájomnej komunikácii jednotlivých častí aplikácie (modulov) a vonkajšou, kedy ide o prepájanie jednotlivých aplikácii do väčších celkov. U oboch týchto integráciách je nutné vykonávať integračné testy.

  Integračné testy sa teda zameriavajú na korektnú komunikáciu jednotlivých modulov, resp. aplikácii. Práve integrácia je z pohľadu vývoja aplikácii kritickou oblasťou. Preto taktiež integračné testy majú svoje nezastúpitelné miesto.

  Integračné testy obvykle postupujú od jednotlivých modulov smerom k väčším celkom. Teda najskorej sú testované rozhrania jednotlivých modulov. Mnohokrát sú v tejto fáze využívané tzv. fake moduly. Ide o simulátory, ktorých úloha je napodobňovať komunikáciu ostatných modulov, ale bez ich funkčnosti. Vďaka nim sa overí, že modul dokáže korektne odosielať i prijímať komunikáciu s ďalšími modulmi.

  Ďalšou fázou je spojovanie modulov do väčších celkov, ktoré majú z pohľadu testovania zmysel. Poslednou fázou je testovanie kompletnej aplikácie.

Z pohľadu integrácie, ktorá sa označuje ako vonkajšia, je nutné vziať do úvahy, že často dochádza k integrácii aplikácii od rôznych výrobcov. Tento druh testovania je tak často náročný na kooperáciu. Musia totiž pri nej spolupracovať rôzne vývojové tímy, nech už je táto spolupráca len na úrovni predania informácii (popis rozhraní) alebo ide o priamu kooperáciu pri testovaní.

- **Systémové testy**

  Pokiaľ je reč o testovaní, najčastejšie sa tým myslia systémové testy. Ide o overenie, že aplikácia ako celok, funguje správne. Testuje sa, či správne plní úlohu, pre ktorú bola vyvinutá, či vracia správne výstupy, či boli ošetrené všetky neštandardné situácie a v neposlednom rade, či boli pokryté všetky požiadavky zo strany zákazníka. Systémové testy obvykle prebiehajú v niekoľkých kolách. Sú hlásené nájdené chyby, tie sú opravované a v nasledujúcich kolách re-testované.

Akceptačné testy

Z pohľadu dodania aplikácie zákazníkovi, sú najpodstatnejšie akceptačné testy. Tie majú overiť, že aplikácia spĺňa zákazníkové požiadavky. Tieto testy môže vykonávať rovnaký tím, ktorý vykonal aj systémové testy, ale rovnako časté je i to, že zákazník poverí týmito testami iný tím, ktorý si často pre tento účel vytvorí. Požiadavky, ktoré zákazník na funkčnosť aplikácie má, je vhodné ešte pred zahájením vývoja (a celkom určite pred zahájením testovania) zhrnúť do tzv. akceptačných kritérii. Tie môžu obsahovať požiadavky na maximálny počet chýb pri testovaní, výkonové a záťažové požiadavky a podobne. Splnením týchto požiadavkov vývoj aplikácie v podstate končí a aplikáciu preberá zákazník.

Testy pre akceptáciu

Nutno povedať, že ani akceptácia nemusí ešte nutne znamenať koniec vývoja aplikácie. Aj keď prebehnú všetky fázy testovania, je stále vysoko pravdepodobné, že aplikácia obsahuje chyby. Preto je v záujme vývojárov tieto chyby opravovať. A s tým súvisí potreba opätov-

ného testovania. Okrem opravy chýb môže dochádzať taktiež k rozširovaniu funkčností aplikácie. V oboch prípadoch je neoddeliteľnou súčasťou vývoja aj testovanie. Opäť sa tu uplatňuje väčšina vyššie spomenutých druhov testov. Nastupuje tu ale i jeden druh testov, ktorý vyššie spomenutý nebol.

Regresné testy

Tieto testy majú za úlohu overiť, že zásahmi do aplikácie nebola narušená správna funkcia tých častí, ktoré týmito zásahmi nemali byť ovplyvnené. Inak povedané, testuje sa, či oprava chyby, alebo pridanie novej funkcionality, nespôsobili novú chybu v už funkčných častiach aplikácie. Regresné testy sa často automatizujú, pretože u nich ide o opakované vykonávanie rovnakých operácii so známym výsledkom. Ich cieľom je tak zistiť, či neexistujú odchýlky medzi výstupom získaným pred zásahom do aplikácie a výstupom po tomto zásahu. Regresné testy nemajú svoje miesto len v okamihu opráv alebo rozvoja už akceptovaných aplikácii. Uplatňujú sa taktiež pri vývoji, ktorý je rozdelený na etapy, ako napríklad agilná metodológia SCRUM. Regresné testy tu overujú, či vývoj v etape neovplyvňuje výstupy z etapy predchádzajúcej. [8]

3.4. Metodológie testovania

Táto sekcia predstavuje metodológie, ktoré budú mať istý súvis s tou prácou. Budú predstavené metódy na modelovanie testovacích prípadov i informácie o kontinuálnej integrácii.

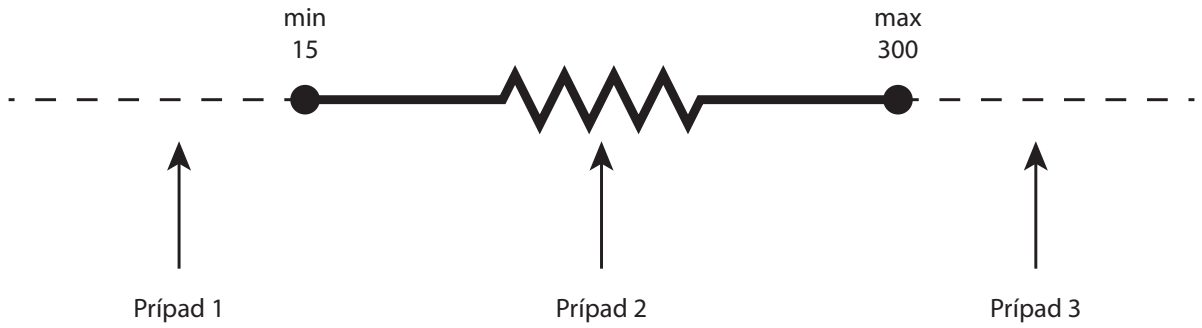
3.4.1. Modelovanie testovacích prípadov

Dizajn testovacích prípadov vyplýva zo zákazníckych požiadavkov a špecifikácie software, ktorá má často formát externej špecifikácie, kde je spísané ako približne by sa mal software správať. Ten, kto bude navrhovať testovacie scenáre, by sa mal na požiadavky pozrieť s nadhľadom, premyslieť možné nástrahy, ktoré môžu z požadovaného správania software vyplynúť a navrhnúť aké výstupy má software vrátiť pri ktorých vstupoch. No pri testovaní software je veľmi obtiažnou úlohou prakticky, niekedy i teoreticky, obsiahnuť všetky

možné prípady, ktoré môžu nastať. Napríklad do textového poľa na webovej stránke môže teoreticky byť zadané takmer nekonečné množstvo kombinácii vstupov. Metódy na modelovanie testovacích prípadov sú vhodné na rozloženie obrovského množstva možných vstupných hodnôt na zopár. Testovanie sa tým pádom stáva efektívnejšie a praktickejšie. [8]

Triedy ekvivalenice

Rozdelenie do tried ekvivalenice je testovacou metódou, vhodnou na riadne ohraničené vstupné hodnoty. Táto metóda by sa dala vysvetliť aj ako rozdelenie vstupných hodnôt do skupín, pričom s každou skupinou sa ďalej počíta ako so samostatnou vstupnou hodnotou. Čiže z danej skupiny hodnôt sa vyberie práve jedna na otestovanie, že daný test bude vyhodnotený správne pre všetky hodnoty z danej skupiny.

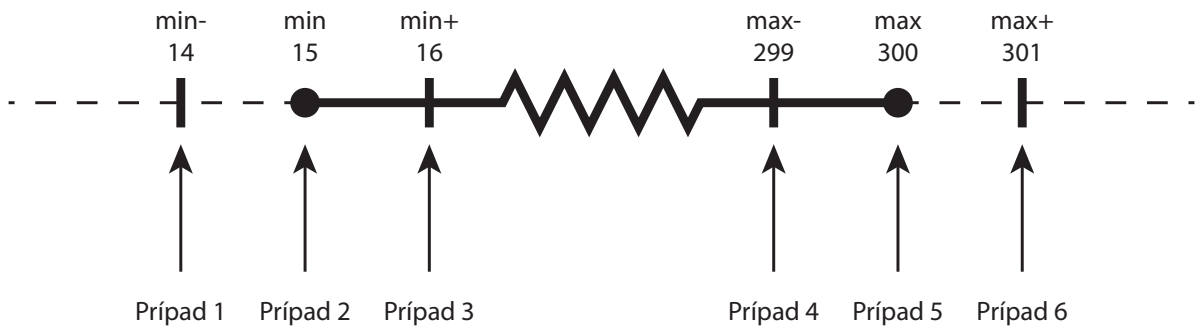


Obr. 3.2: Triedy ekvivalencie (vlastné spracovanie)

V Obrázku 3.2 je zobrazený interval validných vstupných hodnôt v rozsahu od 15 do 300. Všetky možné testovacie prípady sa dajú rozdeliť do troch tried ekvivalencie. Prípad 2 obsahuje validné vstupné hodnoty a ľubovolná hodnota, vybratá z tohto intervalu, bude postačovať pre otestovanie. Prípady 1 a 3 predstavujú intervaly s nižšími a vyššími nevalidnými vstupnými hodnotami. Touto metódou bolo neúnosné množstvo testovacích prípadov rozdelené do troch - výberom hodnôt napríklad **5**, **132**, **376**. [9]

Analýza hraničných hodnôt

Pri testovaní vstupných hodnôt najčastejšie dochádza k chybám pri hodnotách, ktoré sú na okraji testovanej domény. Metódou analýzy hraničných hodnôt, anglicky sa táto metóda značí ako boundary value analysis [40] (BVA) sú testy zameriavané na hodnoty v okolí okrajových bodov testovaného rozsahu. metóda BVA je vylepšenou verziou metódy rozdeľovania na triedy ekvivalencie, nakoľko je kladený vačší dôraz na hraničné prípady.



Obr. 3.3: Príklad analýzy hraničných hodnôt (vlastné spracovanie)

Metódou analýzy hraničných hodnôt je príklad, vyobrazený na obrázku 3.4, rozdelený do šiestich testovacích prípadov. Testovacie prípady sú definované limitnými hodnotami 15 a 300 a ich najbližšími susednými hodnotami. Myers vraví:

„...ak je vykonávaná správne, [analýza hraničných hodnôt] je jednou z najužitočnejších metód na návrh testovacích prípadov.“ [9]

Ďalej pokračuje tvrdením, že náročnou časťou pri využití tejto metódy je práve správne vytýčenie hraničných hodnôt.

Rozhodovacia tabuľka

Rozhodovacia tabuľka je vhodnou metódou v prípade, že kombinácia rôznych vstupných hodnôt, generuje špecifický výstup. Testovanie využitím rozhodovacej tabuľky je výborná

metóda pre vytváranie štruktúr logických členov, ktoré sú intuitívne pochopiteľné pre vývojárov i testerov. Bohužiaľ veľkosť tabuľky, spolu s nárastom vstupných hodnôt, rastie veľmi rýchlo až sa môže stať nepoužiteľnou. Medzi dobré praktiky v tomto prípade patrí rozdeľovať tabuľky na dielčie, najviac ako sa dá.

Podmienka				
Validné užívateľské meno	F	F	T	T
Validné heslo	F	T	F	T

Výsledok				
Chybová správa	x	x	x	
Prihlásenie užívateľa				x

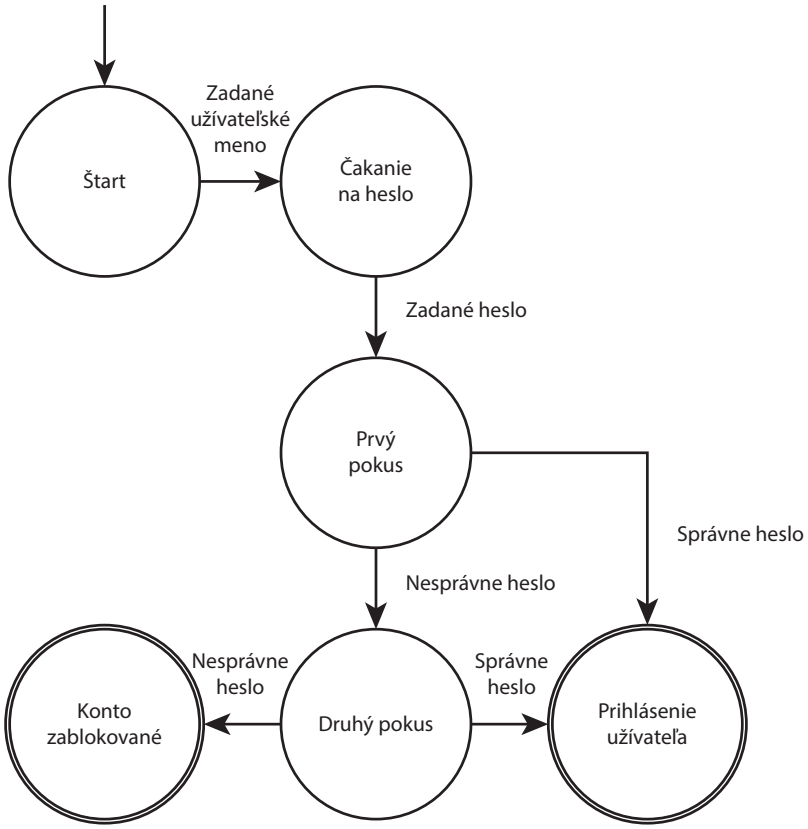
Tabuľka 3.1: Príklad rozhodovacej tabuľky

V tabuľke 3.1 je zobrazená rozhodovacia tabuľka s dvomi podmienkami, z ktorých obe môžu skončiť logickým úspechom (T = True) alebo logickým neúspechom (F = False). Každá kombinácia týchto dvoch ukazateľov produkuje stanovený konkrétny, ktorý by mal byť otestovaný. Vzhľadom k faktu, že tento príklad obsahuje len 4 možné kombinácie, je jednoduché s ním pracovať. S pribúdajúcimi podmienkami však zložitosť tabuľky rastie exponenciálne, podľa vzorca 2^n. Tabuľka je preto nevhodná pre väčšie množstvo vstupov, no pokiaľ je kompletná, poskytuje výborný prehľad všetkých možných kombinácii, ktoré môžu byť inak prehliadnuté. Poskytujú tiež možnosť výberu - ktoré kombinácie budú implementované a ktoré naopak preskočené. [9]

Analýza prechodov medzi stavmi

Väčšina testovacích techník sa zameriava na vyhľadávanie chýb len v jednom izolovanom stave. Analýza prechodov medzi stavmi, anglicky State Transition Analysis - STA [40], sa zameriava na hľadanie chýb pri prechode z jedného stavu aplikácie do druhého. Takéto testovanie v komplexných aplikáciách však býva časovo náročné a náročné i na vykonanie.

Presným definovaním stavov aplikácie a obmedzením možných testových slučiek, môže byť množstvo testovacích prípadov limitované. Na metódu STA sa dá pozreť i na ako konečný automat.



Obr. 3.4: Príklad prihlasovania skrz stavové prechody (vlastné spracovanie)

Pri využívaní metódy STA, môžeme definovať testovaciu podmienku pri každom prechode medzi stavmi systému. Obvykle táto metóda nebýva veľmi efektívna, avšak zo vzniknutého diagramu je jednoduchšie detekovať konkrétne testovacie prípady. [9]

3.4.2. Kontinuálna integrácia

Kontinuálna integrácia CI[3] je jednou z dvanástich základných praktík extrémneho programovania a je správnou voľbou pre priebežné zostavovanie a testovanie vyvíjanej aplikácie. Martin Fowler o CI hovorí:

„ ...vývoj software, kde členovia tímu často integrujú svoju prácu do centrálneho úložiska, pričom každý člen integruje aspoň denne - vedie k mnoho integráciám počas dňa. Každá integrácia si vyžaduje automatické zostavenie, vrátane automatického spustenia testov, aby boli integračné chyby detekované najskôr, ako to je možné.“ [10]

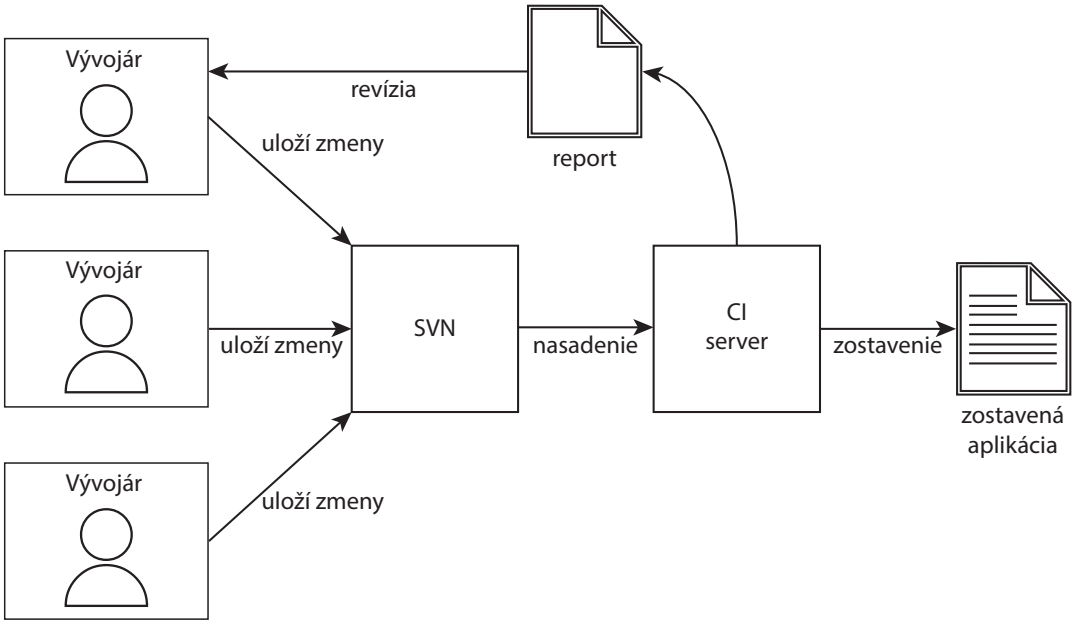
Spúšťanie testov systémom kontinuálnej integrácie nie je povinnosťou, no samozrejme patrí to medzi dobré praktiky, pomáha to odhaliť viac programových chýb. CI po spustení testov vygeneruje report, ktorý sa dostane do rúk developerom a tí môžu na základe neho program doladiť.

„Výsledný produkt, na ktorý bol využívaný systém kontinuálnej integrácie, má tendenciu obsahovať výrazne menej chýb počas vývoja, tak i produkčnom móde.“ [10]

Okrem zvýšenej šance na odhalenie chýb, poskytuje systém kontinuálnej integrácie vývojárom integrovať ich kód kedykoľvek. Vždy, keď je znemená nejaká časť programu, alebo zostavená nová verzia, CI vyhodnotí, či nenastali nejaké integračné chyby. Na nasledujúcom obrázku 3.5 bude ilustrovaná funkcia systému kontinuálnej integrácie.

Prepojením verzovacieho systému (napr. Subversion ako v 3.5) a CI serveru, je server vždy notifikovaný, keď nastanú nejaké zmeny v zdrojových kódoch. CI server následne synchronizuje svoje dáta s repozitárom a vykoná administrátorom definovanú postupnosť

[3] Continuous integration



Obr. 3.5: Kontinuálna integrácia (vlastné spracovanie)

príkazov, medzi ktorými sú i zostavenie novej verzie aplikácie a spustenie automatických testov.

3.5. Agilné techniky testovania - SCRUM

Na začiatok bude zľahka popísaná agilná metodika SCRUM, nakoľko je intenzívne používaná v spoločnosti InQool a zľahka, pretože to nie je cieľom tejto práce.

Základom SCRUMu je tím. Ten je na bežné pomery vývoja software pomerne malý. Tvorí ho tak maximálne desať vývojárov. Role v tíme nie sú striktne rozdelené, preto sa využíva všeobecné pomenovanie člena tímu – vývojár. V tíme by mala fungovať zastupiteľnosť, kedy by ktorýkoľvek člen tímu mal byť schopný zvládnuť ktorúkoľvek činnosť vrámci vývoja software (myslené od analýzy až po testovanie). Jedinou zvláštnou rolou v tíme je pozícia **Scrum Master**. Ten je vlastne z podstaty svojej úlohy trochu mimo tím. Jeho úlohou je pomáhať tímu v úspešnom splnení plánu, ale pritom on sám sa na tomto pláne nepodieľa. Má totiž v popise práce tzv. čistiť tímu cestu a tím administratívne a

organizačne zastrešuje. Moderuje jeho meetingy, pomáha s plánovaním, rieši organizačné a technické problémy (nefunkčný počítač a podobne), odfiltrováva od tímu vonkajšie vplyvy, ktoré by mohli jeho výkon ovplyvniť, ako zamedziť zaúlohovanie člena tímu inou prácou, a taktiež komunikuje s takzvaným **Product Ownerom**. To je ďalšia rola špecifická pre SCRUM. Ide v podstate o zákazníka, ktorý zadáva tímu prácu a následne ju preberá. Stojí mimo tím, ale úzko s ním spolupracuje. Product Owner je ten, kto tímu pripravuje podklady pre plánovanie a následne plán „schvaľuje“.

Plánovanie je z hľadiska SCRUMu prakticky to najdôležitejšie. Plánuje sa v krátkych iteráciách, ktoré sa nazývajú **Sprinty**. Dĺžka sprintu sa v rôznych firmách môže líšiť, ale mala by sa pohybovať od jedného týždňa až po jeden mesiac. Samozrejme záleží na konkrétnych podmienkach na projekte, kde je SCRUM závádzaný. Tím si na nadchádzajúci sprint naplánuje toľko úloh - nazývané ako **User Story**, koľko ich je podľa svojho názoru schopný zvládnuť. Aby tím vedel povedať, koľko práce zvládne za jeden sprint, používa sa odhadovanie. U každej jednotlivej User Story tím odhaduje jej náročnosť. Používajú sa k tomu body, ktoré nemusia mať konkrétnu väzbu na pracnosť v človekohodinách. Tím si skrátka oboduje jednotlivé User Story na základe svojich skúseností a znalostí a to tak, že čím vačší počet bodov, tým náročnejšia úloha.

Aby tím vôbec mohol s plánovaním a odhadovaním začať, musí existovať zoznam požiadavkov, s ktorými môže pracovať. Tomuto zoznamu sa vraví **Product Backlog** a jeho vytvorenie je úlohou Product Ownera. Product Backlog by mal obsahovať všetky požiadavky, ktorými chce daný tím poveriť a ktoré sú v danú chvíľu známe. Nejde teda iba o požiadavky na najbližší sprint, ale malo by byť pokryté oveľa dlhšie obdobie. Na základe týchto požiadavkov tím vytvára svoje User Story, pričom nemusí platiť, že jednému požiadavku odpovedá len jedna User Story.

Vzniknuté User Story tím ohodnotí a následne vyberie tie, ktoré dokáže vrámci sprintu splniť. Tím vzniká **Sprint Backlog**, teda plán na sprint. Toto plánovanie je v rukách

tímu, ktorý musí byť schopný odhadnúť reálny objem práce, ktorý zvládne. Tento objem sa vyjadruje ako rýchlosť, teda počet bodov za sprint. Rýchlosť tímu by mala byť v dlhodobom poňatí plus mínus rovnaká, samozrejme s ohľadom na situáciu v tíme (dovolenky, nemoci a pod.). Vďaka tomu dokáže tím i Product Owner pomerne presne plánovať reálne dokončenie a dodanie výstupov z vývoja. Product Owner môže naviac plánovanie tímu ovplyvniť tým, že jednotlivým požiadavkom v Product Backlogu pridelí priority, čím uprednostní jeden požiadavok pred druhým.

Výsledky svojho snaženia po skončení sprintu prezentuje tím na tzv. **Customer Demo** meetingu. Ide o predvádzanie dokončenej práce, kedy má zákazník možnosť sa k výsledkom vyjadriť a s tímom ich prediskutovať a taktiež naplánovať ďalší postup. Okrem toho sa tím stretáva na tzv. **Stand Up** meetingu, ktorý by mal byť ideálne každý deň a kedy jednotliví členovia tímu prezentujú výstupy svojej práce za predchádzajúci deň a svoju plánovanú činnosť na deň aktuálny.

Jedným z najviditelnejších prejavov toho, že tím funguje v SCRUMe je tabuľa s Sprint Backlogom. Každý tím by mal mať k dispozícii v nejakej forme tabuľu, ktorá zachycuje aktuálnu situáciu pri práci na User Story v danom sprinte. Tabuľa má obvykle tri časti, kde prvou je samotný Sprint Backlog – teda to čo sa má urobiť, druhou časťou je to, čo je už rozpracované - úlohy, na ktorých už niekto pracuje, ale zatiaľ nie sú dokončené a treťou časťou tie úlohy, ktoré už sú dokončené.

Jednou z podstatných informácii je práce to, že v SCRUM tíme by mala fungovať zastupitelnosť a to do tej miery, že by nemali byť rozlíšitelné role ako je vývojár, analytik alebo tester. Príslušnú úlohu, teda i testovanie, by mala vykonávať vždy ten, kto má v danej chvíli čas a to tak, aby sa úlohy jedného typu (napríklad práve testovanie) nehromadily a nečakali na konkrétneho člena tímu. Takto by mal fungovať ideálny SCRUM tím, no v praxi je to ťažko docielitelné. Pri vytváraní tímu sa samozrejme dávajú dohromady ľudia s konkrétnou špecializáciou. Pri fungovaní tímu si potom celkom logicky prednostne

vyberú tie úlohy, ktoré sú im bližšie. čím dlhšie tím funguje, tým viacej sa prejavuje postupné prelínanie rolí. Avšak, stále funguje akýsi status garanta za danú oblasť, teda to, že napríklad o testovaní vie najviac práve tester. [11, 12]

Testovanie a SCRUM – výhody

1. *Rýchla a jednoduchá komunikácia*

   Najväčšia výhoda, ktorú SCRUM testerovi prináša. Tester má priamo v tíme analytika i vývojára a je teda priamo pri zdroji informácii. Môže sa a vlastne aj musí priebežne pýtyť a je pri tom, keď dochádza k rôznym diskusiám o konkrétnych riešeniach alebo prípadných zmenách. Vzhľadom k tomu, že sú všetci členovia jedného tímu a majú spoločný cieľ, odpadá tu akýkoľvek náznak možnej rivality, ktorý sa medzi týmito profesiami objavuje. Nájdenú chybu môže tester okamžite prediskutovať s vývojárom bez toho, že by ju nutne musel predtým zadať do bug-trackingového nástroja. Tým sa veľmi podstatne znižuje riziko zadávania chýb, ktoré sa nakoniec ukážu ako napríklad problém konfigurácie, alebo vstupných dát, prípadne postupu testovania. Ide o skutočne veľkú výhodu SCRUM metodiky. Nedá sa tvrdiť, že by vývojový tím nemohol spolu komunikovať i mimo SCRUM metodiky, no isté je že pokiaľ sú všetci členovia jedného tímu, ktorý sa ako celok zaviazal splniť nejaký objem práce, motivácia jeho členov spolu komunikovať a riešiť problémy je podstatne väčšia ako keď je celý projekt rozdelený na známe etapy (analýza, vývoj, testovanie) a jednotlivé role sú zodpovedné len za splnenie svojej etapy. [11, 12]

2. *Aktívna účasť na hľadaní riešenia*

   Pre testerov, ktorí často fungujú ako upratovacia čata na konci projektu a nemajú príliš šancu ovplyvniť, akým spôsobom je projekt vyvíjaný, ide o vítanú zmenu. V SCRUMe sa ostatní členovia tímu testera proste pýtať musia. Už vo fázi odhadovania a plánovania musí celý tím získať aspoň všeobecnú predstavu o tom, čo otestovanie vyvíjanej funkčnosti znamená a čo bude preto potrebné urobiť. Všetky User Story, i tie zamerané čisto na testovanie ohodnocuje celý tím. Všetci tak musia vedieť o

jak veľký objem práce ide. Naviac ale tester má možnosť hovoriť i do riešenia pri vývoji. Samozrejme je otázka, či jeho pripomienky budú brané do úvahy, no to už záleží na nastavení procesov vnútri tímu. Je ale v záujme celého tímu zapodievať sa tým, či je zvolené riešenie vhodné i z pohľadu testovania. [11, 12]

Tento bod by sa dal nazvať tiež ako „Účasť na projekte od jeho začiatku“. Tester je proste súčasťou tímu po celú dobu vývoja. V SCRUMe sa nestáva, že by bol tester prizvaný do projektu, kedy už je existujúca finálna verzia analýzy a vývojári dokončujú prvú testovateľnú verziu aplikácie. [11, 12]

3. *Dobrá príležitosť k White Box testovaniu*

   V okamihu, kedy je tester priamou súčasťou vývoja a má prístup i k zdrojovým kódom, je veľmi dobrá možnosť posunúť testovanie do roviny White Box. Teda nielen že môže tester konfrontovať reálny kód s predstavou analytika ale môže i pripravovať testy pokrývajúce konkrétne príkazy alebo podmienky v kóde (v duchu White Box). Testovanie tým získa ďalší rozmer a jeho kvalita môže vzrásť.

Testovanie a SCRUM – nevýhody

1. *Nie je k dispozícii finálna verzia analýzy*

   Táto skutočnosť môže predstavovať problém, nakoľko analýza vzniká v rovnakom čase ako kód a taktiež testy. Nie je možné aby pol tímu čakalo, pokiaľ druhá polovica dokončí svoju prácu. SCRUM by strácal zmysel a vznikal by vodopád. Tester tak musí vychádzať z informácii, ktoré získava od analytika a vývojára. To samo o sebe problém nie je. Ten môže nastať až vtedy, keď sa napríklad v polovici sprintu rozhodnú k nejakej zásadnej zmene. Jednoducho povedané vzniká tu nebezpečenstvo nárastu pracnosti vplyvom zmien v zvolenom riešení. S tým musí tím počítať už pri plánovaní. Samozrejme len veľmi zriedka sa stane to, že by nastala tak zásadná zmena, ktorá by úplne znehodnotila doterajšiu prácu testera. Skôr sa môže stať to, že niektoré už pripravené testy stratia svoj zmysel a iné, ktoré neboli v pláne,

sa objavia. Testerovi sa v tejto situácii ťažko reportuje, do akej miery je aplikácia pokrytá testami. [11, 12]

2. *Je zložitejšie naplánovať jednotlivé fázy testovania*

   V SCRUM tíme sa obvykle plánujú User Story ako ucelený blok, ktorý má nejaký prezentovatelný výstup. Súčasťou User Story je tak analýza, vývoj i testovanie. Vzhľadom k tomu, že User Story sa plánujú s ohľadom na schopnosť tímu ho dokončiť v plánovanom sprinte, môže sa stať, že rozčlenenie vývoja aplikácie na tieto úseky nemusia odpovedať tomu, ako by testovanie aplikácie naplánoval tester, keby fungoval samostatne. Môže sa vyskytnúť napríklad potreba dielčích testov, ktoré by inak neboli v tejto podobe vykonávané a boli by súčasťou nejakých väčších testovacích scenárov. Tester v tomto musí rešpektovať potreby tímu a fakt, že na konci sprintu tím odovzdáva svoju prácu, ktorá by mala byť, okrem iného, otestovaná. Na druhú stranu, tím by mal zas brať do úvahy testovateľnosť vyvíjanej a dodávanej časti aplikácie. [11, 12]

3.6. Podnikové procesy

Systém manažmentu kvality zahŕňa oblasť manažmentu procesov, ktorej význam stále rastie najmä z dôvodov neustálych zmien a flexibility na trhu, čo vedie k pokusom o neustále zlepšovanie procesov. Procesný prístup riadenia spoločnosti je založený na vzájomnom pôsobení jednotlivých procesov tak, aby napĺňali určené ciele. Efektívne prebiehajúce procesy sú nástrojom udržania a posilnenia pozície podniku v trhovo-orientovanom prostredí konkurencie a tak isto majú kladný vplyv i na výšku nákladov, objem tržieb a teda aj na generovaní zisku. Preto je dôležité, aby podniky venovali pozornosť organizácii vlastných procesov a snažili sa o ich neustále zdokonaľovanie. Systém procesného riadenia (angl. Business Process Management) je efektívnym nástrojom na zabezpečenie prevádzkovej efektívnosti pri súčasnom napĺňaní požiadavkov zákazníka i vnútorných potrieb spoločnosti. Tento systém umožňuje spoločnosti efektívne navrhnúť podnikové procesy, zoptimalizovať

väzby medzi nimi a tiež vytvoriť organizačnú štruktúru, ktorá bude efektívnosť priamo podporovať. [13]

Proces ako pojem

V súčasnosti existuje mnoho rôznych pohľadov na definíciu pojmu proces, či procesné riadenie.

„Proces je súbor previazaných činností, ktoré vezmú vstup, transformujú ho a vytvoria výstup.“[18]

„Proces je definovaný ako organizovaná skupina vzájomne súvisiacich činností a / alebo subprocesov, ktoré prechádzajú jedným alebo viacerými organizačnými útvarmi či jednou alebo viacerými spolupracujúcimi organizáciami, ktoré spotrebúvajú materiálne, ľudské, finančné a informačné vstupy a ich výstupom je produkt, ktorý má hodnotu pre externého alebo interného zákazníka.“ [17]

3.6.1. Procesný prístup riadenia

Základnou črtou procesného riadenia je možnosť rýchlej reakcie a naplnenie rôznych typov zákazníkových potrieb. Tento prístup dopomáha vytrávať pohľady na spoločnosť z rozličných uhlov, popisuje spoločnosť skrz prebiehajúce činnosti a dovoľuje zvyšovať efektivitu, hospodárnosť a účelnosť prebiehajúcich výkonov a procesov v organizácii.

Organizácie sa navzájom líšia hlavne spôsobom, ako sú v nich jednotlivé procesy vykonávané a riadené, čo má priamy dopad na celkovú efektívnosť organizácie. V prípade, že organizácia kladie svojím spôsobom riadenia “odpor” interne prebiehajúcim procesom – či už vo forme neproduktívnej internej komunikácie, zbytočnej dokumentácie, zbytočného zasahovania veľkého počtu organizačných útvarov a pracovníkov a podobnej byrokracie – zvyšuje tým čas a náklady na priebeh procesov, znižuje kvalitu výstupov a spokojnosť

zákazníka. Pre udržanie konkurencieschopnosti organizácie je nutné prispôsobiť systém riadenia a organizačnú štruktúru iným procesom tak, aby bolo možné procesy priamo riadiť, kontrolovať a stanoviť pre ne meradlá výkonnosti a mať možnosť neustále zlepšovať výkonnosť spoločnosti. Pretože čo nie je merateľné, to nie je možné ani riadiť. [14]

3.6.2. Rozdelenie podnikových procesov

Moderná organizácia má vytvorený, zdokumentovaný, zavedený a udržiavaný systém manažmentu kvality v súlade s požiadavkami normy ISO 9001:2008 a trvalo pracuje na zefektívňovaní procesov. Procesy identifikované a riadené vrámci organizácie sú rozdelené do troch hlavných kategórii, vzhľadom na ich vzťah k dosahovaniu kvality výsledných produktov a služieb a ich vzájomnej interakcii:

- *Manažérske procesy*

  Slúžia na podporu podnikových procesov a primárne súvisia s úspešným fungovaním podniku. Patrí sem napr. strategické plánovanie, interná/externá komunikácia, riadenie ľudských zdrojov či spokojnosť zákazníkov. Prostredníctov týchto procesov je v spoločnosti riadený tok informácii potrebný pre fungovanie celého systému procesov a plánovanie vo všeobecnosti.

- *Hlavné procesy*

  Jedná sa o zásadné procesy pre fungovanie podniku, napĺňajúce samotný účel podnikania – súbor činností, ktoré vedú od požiadavkov zákazníka, skrz implementáciu, až po ich uspokojenie a ďalej vedú k následnému uhradeniu produktu či služby. Príkladom je riadenie dopytu, riadenie zakázky, plánovanie a riadenie výroby, expedícia, fakturácia...

- *Podporné procesy*

  Sú to procesy, ktoré sú pre zákazníka „neviditeľné“, no z hľadiska fungovania spoločnosti sú nevyhnuté a ich vplyv na klavitu služby či výsledného produktu je veľký. Môžu byť zabezpečené i externe, avšak z dôvodov minimalizácie rizík, či kvôli ekono-

mickej efektívnosti často bývajú aplikované interne. Napríklad realizácia serverového úložiska skrz vlasné servery, alebo prechod do cloudu. [15]

3.6.3. Zásady procesného riadenia

Z hľadiska fungujúceho systému procesného riadenia je dôležité, aby sa spoločnosť zamerala na identifikáciu svojich vlsntných procesov, ďalej aby dokázala verifikovať činnosti vedúce k premene vstupov na výstupy a v neposlednom rade neustále tento systém monitorovať, merať a neustále zlepšovať. Grasserová [16] špecifikuje desať princípov procesného riadenia:

1. **Integrácia a kompresia prác**

   Integrácia práce do logických celkov a súčasne zhustenie prác, pričom zbytočné činnosti sa vylúčia

2. **Delinearizácia prác**

   Vykonávanie prác v prirodzenom slede

3. **Najvýhodnejšie miesto na prácu**

   Vykonávanie prác na najvýhodnejšom mieste, bez ohľadu na oddelenie, útvar či podnik

4. **Uplatnenie tímovej práce**

   Zaisťovanie procesov skrz autonómne tímy s dostatočnými kompetenciami a motiváciou

5. **Procesne zameranie motivácie**

   Napojenie motivácie i na výsledný produkt, nielen na činnosť

6. **Zodpovednosť za proces**

   Určenie vlastníka procesu, ktorý je zodpovedný za jeho efektivitu a ďalšie zdokonaľovanie

7. **Variantné chápanie procesu**

   Pohľad na proces z viacerých uhlov, neexistuje len jedna varianta procesu a zvolenie správenho prístupu záleží na okolnostiach

8. **3S**

   Posilnenie autonómie tímu, pričom tím je samostatná jednotka, schopná plniť určené ciele, 3S = samoriadenie, samokontrola, samoorganizácia

9. **Pružná autonómia procesných tímov**

   Zostavenie tímu tak, aby bol maximálne flexibilný

10. **Znalostná a informačná bezbariérovosť**

    Využitie napr. znalostných systémov k šíreniu znalostí a informácii skrz organizáciu

3.6.4. Business Process Management

Business process management, skrátene BPM, by sa dal voľne definovať ako systematická snaha ovplyvniť správanie konkrétneho subjektu tak, aby bolo predvídateľné, správne reagovalo na podnety a poskytovalo úžitok tomu, pre koho sa daná činnosť vykonáva. Tento koncept znamená prevratnú inováciu vo využívaní informačných technológii na podporu firemných procesov. Hlavným kladom tohto konceptu je, že pri jeho využívaní nie je treba prihliadať na typ, odvetvie na ktoré sa zameriava, či veľkosť spoločnosti. Princípom nie je len inovácia pôvodných procesov, ale hlavne realizovať a riadiť celý prúd prebiehajúcich procesov, ktoré vedú k naplneniu strategických cieľov. Vrámci BPM sú procesy hlavným prostriedkom odlíšenia sa od konkurencie a intelektuálnym vlastníctvom spoločnosti. Význam metodiky BPM je založený na fakte, že na proces nadefinovaný v BPMN notácii, sa dá pozrieť z rôznych uhlov a zároveň interpretovať s istou odlišnosťou. [19]

3.7. Mapovanie a modelovanie procesov

Procesy sa dajú zjednodušene pomenovať ako sekvencie aktivít nadizajnované tak, aby transformovali vstupné dáta na výstupné. Procesné mapovanie v spoločnosti má za úlohu identifikovať prebiehajúce aktivity, zanalyzovať ich a výsledky využiť na lepšie porozumenie firemných procesov a ich prípadnú optimalizáciu. K uchopeniu zmyslu procesu slúži tzv. procesná mapa, ktorá sa skladá z hierarchicky systematizovaných grafických diagramov. Procesné mapovanie zachytáva:

- toky materiálu, informácií a dokumentov
- najrôznejšie úlohy, ktoré prebiehajú vnútri mapovaného procesu
- ako proces transformuje vstupy na výstupy
- rozhodnutia, ktoré musia byť vrámci procesu vykonané a v akom poradí

Modelovanie firemných procesov je činnosť, ktorej náplňou je tvorba modelu firemného procesu, ktorý vznikne ako abstrakcia reálneho podnikového procesu. Technika modelovania záleží na výbere konkrétneho nástroja a jazyka. Každý jazyk má difonovanú syntax a sémantiku, je dôležité aby človek čo modeluje procesy tieto záležitosti ovládal. Vo všeobecnosti platí, že každá úloha sa dá namodelovať ako proces, samotná zložitosť už záleží na nástroji, autorovi, zrozumiteľnosti, obmedzeniach a nie na samotnom obsahu modelovaného procesu. Notácia sa dodnešku veľmi rozšírila a je využívaná veľkým množstvom nástrojov. Dnes sa BPMN považuje za štandard modelovania podnikových procesov. [15]

3.7.1. Business Process Management Notation

Notácia BPM predstavuje súbor grafických objektov a pravidiel, ktoré slúžia ako prostriedok na modelovanie procesov. Jej vznik bol iniciovaný organizáciou Business Process Management Initiative, ktorej primárnym cieľom bolo vytvoriť notáciu, ktorá bude jednoducho čitateľná a pochopiteľná pre všetkých stakeholderov od vrcholového managementu, ktorí budú daný proces do organizácie zavádzať, cez technických inžinierov, ktorí ho budú

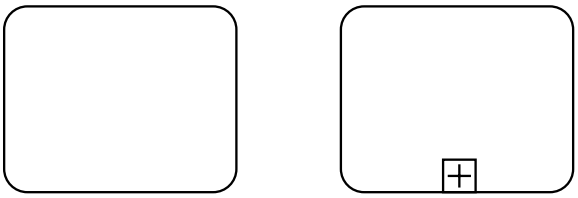
napĺňať, až po analytikov, ktorí ho budú manažovať a monitorovať. Nasledujúca časť práce obsahuje popis základných grafických objektov, ktoré budú ďalej využité pri modelovaní. [20]

Udalosť (Event)

Obr. 3.6: Udalosť (vlastné spracovanie podľa [20])

Udalosť prezentuje v procesi jav, ktorý ho priamo ovplyvnil. Delia sa na počiatočné, priebežné a konečné. Obrázok 3.6 zobrazuje počiatočnú udalosť (vľavo), priebežnú so správou (v strede) a konečnú (vpravo). [20]

Aktivita (Activity)



Obr. 3.7: Aktivita (vlastné spracovanie podľa [20])

Aktivita predstavuje krok v procesi, ktorý zahŕňa určitú činnosť alebo prácu. Môže byť atomická, na 3.7 vľavo, alebo komplexná, ktorá zahŕňa celý subproces, na obrázku 3.7 vpravo. [20]

Brána (Gateway)

Brána predstavuje bod vetvenia alebo zbiehania procesov. Na obrázku 3.8 vľavo je exkluzívna brána, z ktorej proces pokračuje ďalej len jednou cestou na základe vyhodnotenia

Obr. 3.8: Brána (vlastné spracovanie podľa [20])

podmienky. Vpravo na obrázku 3.8 je paralelná brána, ktorá rozdeľuje proces do subprocesov, ktoré bežia ďalej paralelne. [20]

Sekvenčný tok (Sequence flow)

Obr. 3.9: Sekvenčný tok (vlastné spracovanie podľa [20])

Sekvenčný tok predstavuje sekvenciu aktivít, teda ako po sebe nasledovať. Zdrojom a príjemcom musí byť udalosť, aktivita alebo brána. [20]

Tok správ (Message flow)

Obr. 3.10: Tok správ (vlastné spracovanie podľa [20])

Tok správ zobrazuje tok správ medzi dvomi objektami. [20]

Bazén/dráha (Pool/lane)

Bazén predstavuje účastníka procesu alebo entitu a zobrazuje tok správ medzi viacerými účastníkmi. Môže byť použitý i ako *black-box* prvok, kedy nemá žiadny obsah. Podmnožinou bazénu je dráha, ktorá delí bazén na viacero entít. [20]

Obr. 3.11: Bazén/dráha (vlastné spracovanie podľa [20])

Artefakty (Artifacts)

Obr. 3.12: Artefakty (vlastné spracovanie podľa [20])

Artefakty pridávajú BPM diagramu informačnú hodnotu, no do samotného priebehu procesu nezasahujú. Patria sem elementy ako dátové objekty, ktoré sa využívajú na zobrazenie dokumentov vnútri procesu (3.12 vľavo). Ďalej skupiny, ktoré združujú súvisiace elementy do jedného celku (3.12 v strede) a nakoniec poznámky, na obrázku vpravo. [20]

4. Analýza súčasného stavu

V tejto kapitole bude rozobratý súčasný stav dostupných technológii a nástrojov, ale aj rozbor procesov spoločnosti InQool a.s., do ktorej budú navrhnuté riešenia neskôr implementované.

4.1. Spoločnosť InQool a.s.

Spoločnosť InQool a.s. je mladá softwarová spoočnosť, ktorá bola založená v roku 2011. Hlavným podnikateľským zámerom je tvorba komerčných riešení v rôznych sférach ako napríklad bankovníctvo, komunikácia, financie.

Interný spôsob fungovania

Ku dňu 7.5.2016 má spoločnosť 19 stálych zamestnancov, ktorí tvoria jej gro. Vekové rozloženie je od 21 do 29 rokov, čo znamená, že vedenie sa zameriava na výber mladých, schopných ľudí do kolektívu a vnútorným prístupom si chce zabezpečiť ich lojalitu. Spoločnosť je vedená moderným spôsobom - zo spoločnosti sa nesnažia vytvárať korporát, naopak vyhýbajú sa byrokracii, neexistuje žiadna fixná hierarchia ani ďalšie zbytočné firemné štruktúry. InQool vnútorne funguje ako startup, procesy na vývoj software sú navrhnuté agilne, dôraz je kladený na človeka a jeho individuálny rozvoj. V spoločnosti je zastávaná idea, že ak sa zamestnanec cíti pohodlne, odvádza lepšiu prácu. Zamestnanci sú vrámci spoločnosti rozdelení do väčších či menších tímov. Každý tím udržiava alebo vyvíja jeden či viacej projektov. Niektoré projekty si vyžadujú viac pozornosti, niektoré menej. Spravovaných projektov je momentálne 18.

Z uvedených faktov vyplýva, že v spoločnosti panuje neformálna atmosféra, vývoj software nasleduje princípy agilnej metodiky SCRUM, čiže návrh procesov QA sa bude niesť v podobnom duchu, aby neboli fungujúce procesy narušené, ale aby i naďalej fungovali s nálepkou, ktorá bude zaručovať ich kvalitu.

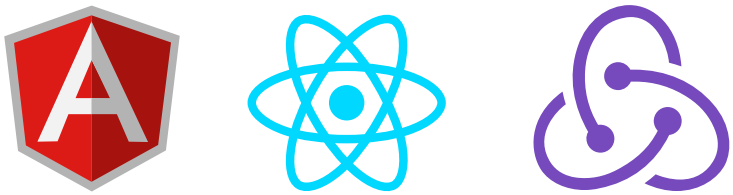
Doterajšie QA

Pojmu Quality Assurance doteraz v spoločnosti nebol kladený veľký význam. Explicitná pozícia testera neexistovala, software bol vyvýjaný programátormi, následne nimi i manuálne otestovaný a nasadený do produkčného módu. Tento spôsob je však dlhodobo neudržateľný, hlavne ak sú projekty rozsiahle a závislé na fakte, že všetko bude fungovať správne. Z času na čas sa stalo, že zákazníkovi bola odovzdaná nefunkčná časť software. Dôvodom je práve nedostatočné otestovanie. Testovanie software bolo zavedené len na backendovej časti informačných systémov, i to len niektorých, formou jednotkových testov (popísané v sekcii 3.3). Cieľom tejto práce je preto rozšíriť testovanie v spoločnosti i na frontend časť systémov.

Využívané technológie v spoločnosti

Väčšina informačných systémov, ktoré sú vyvíjané spoločnosťou InQool, je rozdelená na backend a frontend časti. Backend tvorí funkciu RESTful API[1], vyvinutý prevažne v programovacom jazyku Java, prípadne PHP. Ako úložisko pre dáta slúži prevažne PostgreSQL databáza.

Frontend aplikácii je budovaný najmodernejšími technológiami, založenými na programovacom jazyku JavaScript, ktorý sa v posledných rokoch teší veľkému rozmachu.



Obr. 4.1: Logá AngularJS a ReactJS a Redux technológii [22, 23, 24]

[1]REST (Representational State Transfer) je architektúra, ktorá umožňuje pristupovať k dátam na určitom mieste pomocou štandardných metód HTTP - POST, GET, DELETE, PUT.

1. **React**

   React je JavaScriptová knižnica na budovanie užívateľských rozhraní, ktorá pochádza priamo od vývojárov sociálnej siete Facebook a Instagram. Nie je to komplexný framework, mnohými je považovaný za View v architektúre MVC[2]. Základná myšlienka, s ktorou bol React vyvinutý, je: *Budovanie rozsiahlych aplikácii s dátami, ktoré sa v priebehuh času menia.*

   React je charakteristický tým, že preferuje vkladanie HTML tagov do JavaScriptového kódu. Táto vlastnosť má však okrem prívržencov i striktných odporcov, ale či je to návrh korektný alebo nie, je na samostatnú diskusiu. Oproti konkurencii má hlavnú výhodou v tom, že preklad celej aplikácie môže prebehnúť na serveri, čím odpadá problém s indexovaním vo vyhladávačoch. [22]

2. **Redux**

   Redux je predikovateľný stavový kontainer pre aplikácie programované v JavaScripte. V princípe sa nejedná o programovací jazyk alebo framework, ide o architektonický návrh, ktorý podporuje konzistentný beh aplikácie. Je ho možné využívať v kombinácii s ľubovoľnou knižnicou na budovanie užívateľských rozhraní, v spoločnosti InQool sa používa v kombinácii s ReactJS, ktorá je odporúčaná aj v oficiálnej dokumentácii. Táto architektúra bola vytvorená Danom Abramovom v roku 2015, spočiatku ako malý projekt, no svojimi myšlienkami si rýchlo získal mnoho podporovateľov. [23]

3. **Angular**

   AngularJS je populárny MVC framework, ktorý má za sebou dlhšiu históriu, ako len v posledných rokoch vyvinutý React. Medzi jeho hlavné charakteristiky patrí dvojcestná synchronizácia dát[3], implementácia Dependency Injection[4], testovateľnosť, direktívy či znovupoužiteľnosť komponent. Momentálne sa v InQoole vývoj fron-

[2]Model-View-Controller

[3]zaužívaný je anglický pojem Two Way Data-Binding

[4]návrhový vzor, ktorý rieši závislosti medzi jednotlivými komponentami programu

tendu uberá skôr orientáciou na React, no zopár projektov je aktívne udržiavaných a vyvíjaných práve i v tejto technológii. [24]

Kontinuálna integrácia nástrojom Jenkins

Medzi najpopulárnejšie servery kontinuálnej integrácie, ktoré sú voľne dostupné patrí Jenkins, ktorý je využívaný aj v spoločnosti InQool. Jedná sa o rozšírenie CI servera Hudson, vybudovaného na Java platforme. [25] Zamestnanci v spoločnosti majú udelené práva k jednotlivým projektom podľa potreby, autorizácia prebieha skrz LDAP[5] protokol. K tomuto serveru je dostupných mnoho rozšírení, niektoré sú využívané i v spoločnosti InQool. CI server podporuje spoluprácu s revíznym systémom, čo znamená, že ihneď po nahratí zmeny do repozitára, je vykonaný skript. Takto je nakonfigurované zostavenie, testovanie backendu i nasadenie aplikácie. Princíp priebehu kontinuálnej integrácie v spoločnosti InQool je totožný s vizuálizáciou v teoretickej časti 3.5. Úlohu úložiska Subversion, ktorý je zobrazný v diagrame, v spoločnosti plní Gitlab, čo je webový software na správu Git repozitárov.

V systéme Jenkins je možné nastaviť prostredie pre spracovanie výsledkov testov, či už regresných, jednotkových či integračných. To je veľká výhoda systému, nakoľko výsledky testov väčšinou predstavujú len štruktúrované dáta, napríklad vo formáte XML. Jenkins obsahuje veľké množstvo rozšírení a medzi nimi i nástroj na vizuálizáciu štruktúrovaných dát.

4.2. Projekt Artstaq

Metodiky testovania budú navrhované a implementované do tohto projektu, preto bude v tejto časti uvedený rozbor súčasného stavu projektu.

Artstaq STANDARD je skratka pre Automated Rating & Trading Standard for Art Qu-

[5]LDAP je protokol pre prístup a správu distribuovaných adresárových služieb.

otation, štandard pre posilnenie obchodovania s umením pre skúsených obchodníkov, ako aj pre nováčikov medzi investormi. Artstaq je relatívne nový startup, ktorý pripravuje pôdu pre investície do umenia, ktoré má momentálne dve hlavné úskalia:

- nedostatok dôvery voči investíciám do výtvarného umenia, z dôvodu neprehľadných trhov
- deficit princípov obchodovania v umení, ktoré nepostrádajú regulačné normy

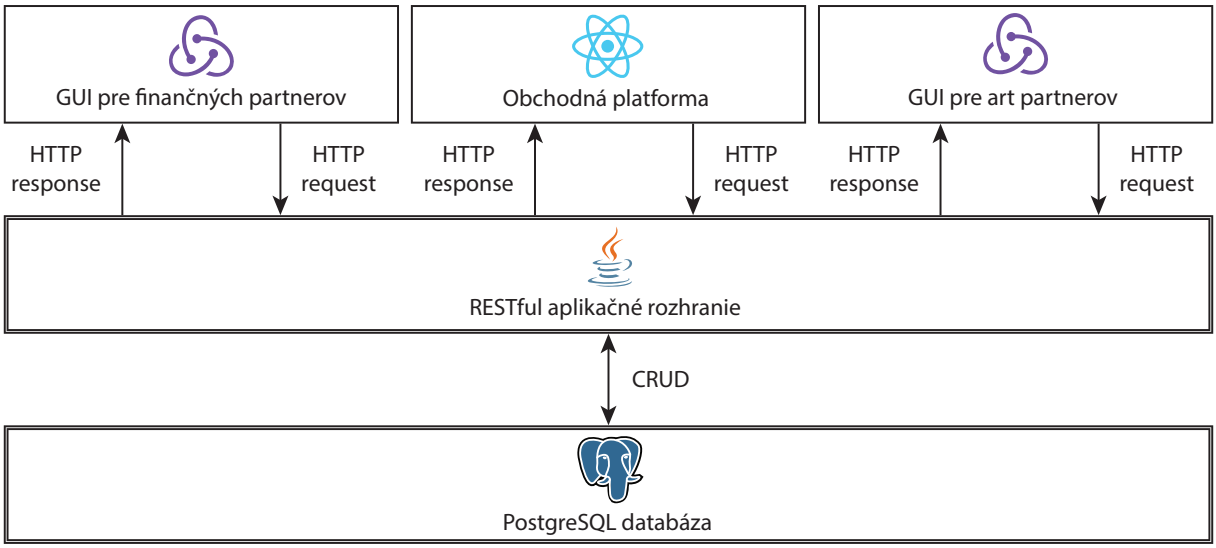
Tento štandard definuje rámec, ktorý zvyšuje kvalitu, integritu a efektivitu obchodovania skrz platformu Artstaq Exchange. Jedná sa o investovanie do umenia za cieľom zhodnotenia majetku, podobne ako to funguje s cennými papiermi, zlatom a podobne.

Takto vyzerá vyčerpávajúca definícia projektu, na základoch ktorého je postavený informačný systém burzového typu. Informačný systém stojí na dvoch platformách - Artex500 (globálny trh - https://artex500.com/) a Bolsaarte (trh Južnej Ameriky - https://bolsaarte.com/). Tieto obchodné platformy sú cloudovým riešením typu Saas pre registrovaných užívateľov, ktorí môžu v reálnom čase obchodovať s umeleckými dielami. Ďalej je pre finančných partnerov poskytované samostatné užívateľské rozhranie pre administráciu svojich investorov a samostatné pre Art partnerov, ktorí majú na starosti administráciu umelcov a ich diel v systéme.

Technická stránka projektu

Z technologického hľadiska sa obchodná platforma skladá z troch samostatných javascriptových aplikácii, ktoré zdieľajú spoločný server s jednou databázou. Aplikácie sú samostanté pre rozhranie investorov, finančných partnerov a Art partnerov. Aplikácie sú vyvinuté za využitia JavaScriptovej knižnice React a architektonického rozšírenia Redux.

V čase, kedy boli metodiky testovania navrhované a implementované bol už systém do



Obr. 4.2: Architektúra systému Artstaq (vlastné spracovanie)

značnej miery vyvinutý, čo nepodlieha *best pratices*, že testovanie by malo byť vykonávané počas všetkých etáp životného cyklu software. Do budúcnosti to však ukazuje priestor, v ktorom by sa mala spoločnosť InQool zlepšiť.

Štatistiky aplikácii

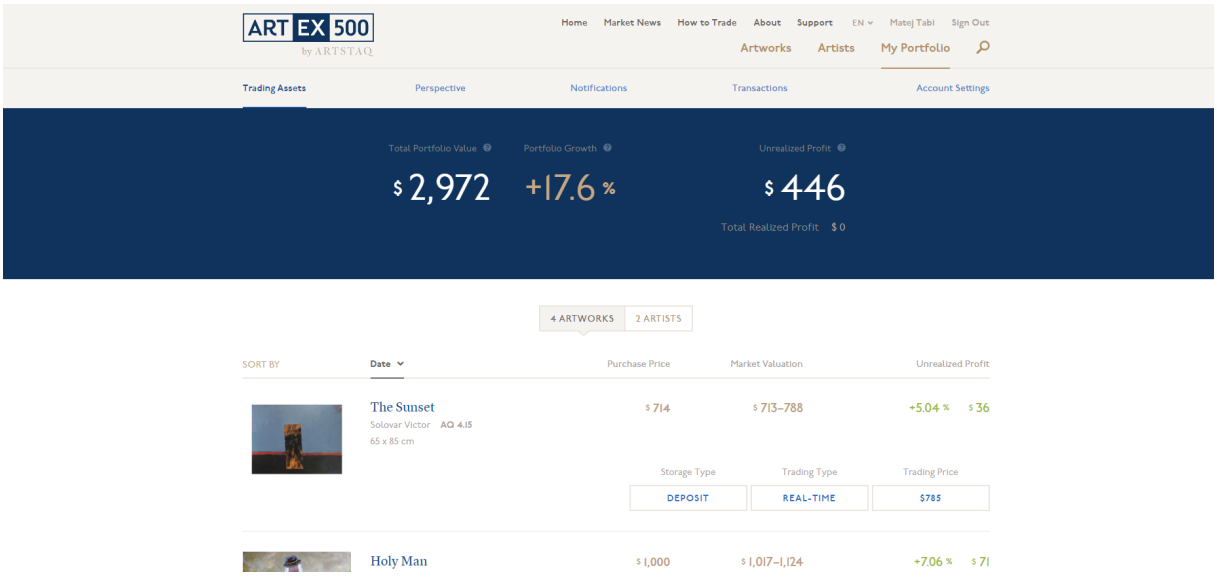
Z uvedených štatistík vyplýva, koľko práce bude s jednotlivými časťami systému pri návrhu testovacích scenárov a so samotným pokrytím aplikácii testami.

	Trade Platforma	Fin Partner Backoffice	Art Partner Backoffice
Počet súborov	932	175	228
Počet riadkov kódu	603 310	133 233	133 319
Počet užívateľských scenárov	104	19	63

Tabuľka 4.1: Robustnosť aplikácii

Najrobustnejšia časť je jednoznačne Trade platforma pre investorov. Platí približná rovnosť, že v každom súbore sa nachádza samostatná komponenta, na ktorú sa dajú zvlášť napísať jednotkové testy [3.3]. Partneri majú možnosť vo svojich užíateľských rozhraniach

generovať tzv. membership karty. Tieto karty slúžia ako pozvánka investorovi na vstup do systému Trade platformy. No do obchodnej platformy sa môže užívateľ i registrovať priamo, čiže v konečnom dôsledku tam má povolený prístup ktokoľvek. Tým pádom musí byť obchodná platforma pripravená na nápor užívateľov a každý má k dispozícii vlastné rozhranie, kde mu je zobrazené jeho portfólio umeleckých diel a rôzne nastavenia, s kladením dôrazu na prehľadnom a moderné UX (viz priložený screenshot 4.3).



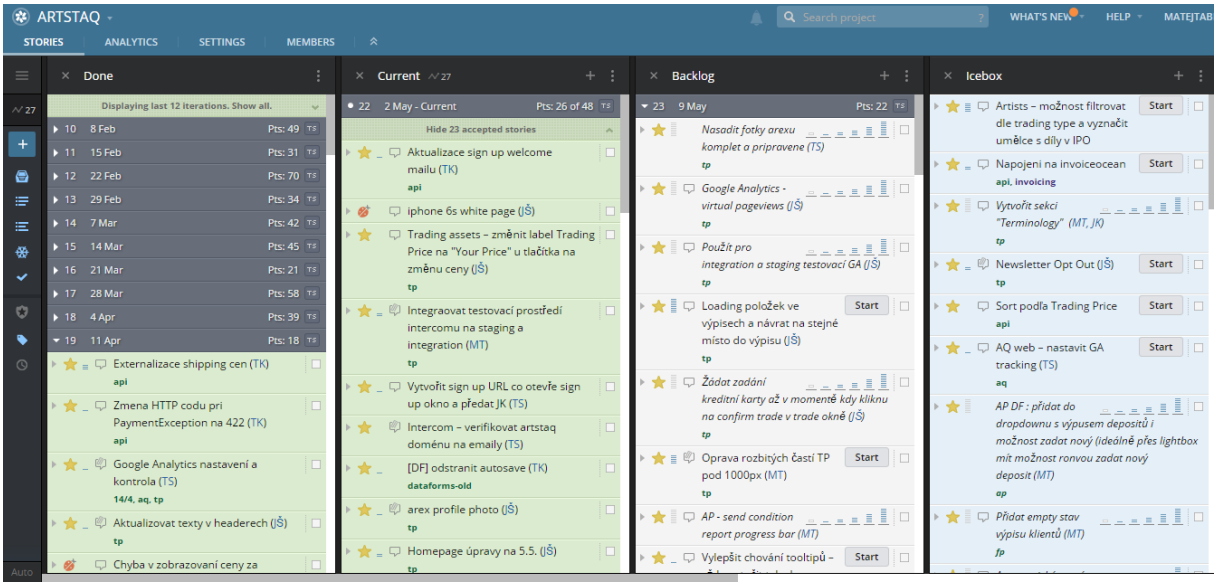
Obr. 4.3: Užívateľské rozhranie investora [41]

Ogranizácia nástrojom Pivotal Tracker

Na projekte je z pohľadu vývojového procesu implementovaná agilná metodika SCRUM, tak ako je popísaná v teoretickej časti práce. Sprinty sú rozdelené do dvojtýždňových cyklov, v prípade akútnych úprav jednotýždňových. Tím pracuje ako samostatná jednotka, dôraz je kladený na dobré medziľudské vzťahy, všetci členovia pracujú v jednej kancelárii, čo výrazne zlepšuje komunikáciu. Náplne práce sa navzájom prelínajú, čo spôsobuje že medzi jednotlivými rolami nevzniká jav súperenia, ale všetci ťahajú *za jeden koniec* a navzájom si pomáhajú.

Na organizáciu práce je využívaná online služba Pivotal Tracker [42], kde je vytvorený Dashboard pre projekt Artstaq a prístup k nemu majú všetci členovia tímu. Táto služba poskytuje všetky potrebné záležitosti pre ideálnu organizáciu vývojového procesu.

Ako bolo uvedené v teoretickej časti i vrámci projektu Artstaq prebieha intenzívna komunikácia medzi vývojovým tímom a Product Ownerom. Na začiatku životného cyklu je pridaný konkrétny User Story do Product Backlogu (Iceboxu). Pivotal tracker je nástroj, poskytujúci istú úroveň automatizácie, čiže do konkrétneho sprintu automaticky vyberá zvládnuteľné množstvo User Stories, vychádzajúc pri tom zo štatistiky zapracovaných požiadavkov počas minulých sprintov. Po výbere User Stories do konkrétneho sprintu vzniká Sprint Backlog a úlohou Scrum Mastera je priradenie ku každému zákazníckemu požiadavku vývojára, ktorý má na starosti danú časť systému. Vývojár úlohe pridelí body, ktoré má tím stanovené a zaužívané. Tieto body sa udeľujú na základe náročnosti zapracovania, čím náročnejšia úloha, tým viacej bodov.

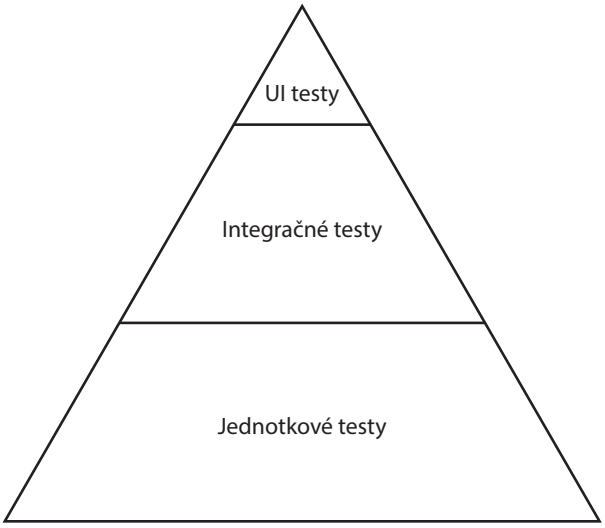


Obr. 4.4: Dashboard projektu Artstaq [42]

Po skončení implementácie požiadavku je nahraná zmena do Git úložiska, kde je integračným prostredím Jenkins zostavená nová verzia aplikácie, následne nasadená na integračné prostredie (testovacie). Po otestovaní Product Ownerom je zákaznícky požiadavok - ktorý je momentálne v stave finished - akceptovaný a vrámci systému Pivotal Tracker je presunutý do sekcie Done.

4.3. Technológie testovania JS

V tejto sekcii bude uvedený prierez dostupných technológii testovania JavaScriptu. Pri výbere bude zohľadňovaný profil spoločnosti InQool tak, aby zvolené technológie nasledovali existujúce strategické ciele, ale zároveň aby dodržiavali koncept pyramídy automatizovaného testovania, zobrazeného na obrázku 4.5 [26].



Obr. 4.5: Koncept pyramídy automatizovaného testovania (podla [26])

Sekcia bude rozdelená na tri podsekcie, pričom v prvej bude popis technológii pre jednotkové testovanie, v druhej zase pre vykonávanie integračných testov a najmenšiu časť budú tvoriť technológie pre automatizovanie UI testov.

4.3.1. Technológie pre jednotkové testy

Tak ako mnoho ďalších vecí vo svete JavaScriptu, existuje viacero prístupov k jednotkovému testovaniu JavaScriptového kódu. Všetky uvedené nástroje dobre slúžia svojmu účelu a udržujú si aktívne komunity. Od roku 2012 však JavaScript prechádza masívnym rozmachom, preto toto porovnanie platí pre dnešok, o rok už môže byť neaktuálne. [21]

QUnit

Testovací framework `QUnit` [27] bol pôvodne vyvinutý Johnom Resigom ako súčasť frameworku `jQuery`. V roku 2008 bol od `jQuery` oddelený, vznikla dokumentácia, čo umožnilo jeho využívanie na jednotkové testovanie JavaScriptu vo všeobecnosti, no v tejto dobe ešte stále existovali závislosti na `jQuery`. Od roku 2009 bol `qUnit` úplne oddelený od `jQuery` a odvtedy existuje ako úplne samostatný framework.

Dnes patrí k populárnym JavaScriptovým frameworkom na jednotkové testovanie. I keď je `QUnit` využívaný prevažne na testovanie `jQuery`, môže byť použitý aj na testovanie ľubovoľného JavaScriptového kódu. `QUnit` poskytuje jednoduchú syntax na vytváranie testových modulov a funkcii, ktoré môžu byť spúšťané z prehliadača.

Medzi jeho výhody patrí jednoduchosť použitia, či voľba ľubovoľného poradia spúšťaných testov. Nevýhodou je chýbajúca podpora source máp[6] alebo komplikovaný spôsob testovania asynchrónnych konštrukcii.

```
QUnit.test("hello test", function(assert) {
  assert.ok(1 == "1", "Passed!");
});
```

Snippet kódu 4.1: Snippet testu napísaného v `QUnit` [27]

[6] využíva sa na debuggovanie, kedy je produkčný debuggovaný kód mapovaný do originálnych zdrojových kódov

Framework je vhodné využiť, pokiaľ je testovaná aplikácia malá a všetky vykonávané testy budú prebiehať na strane klienta. Pokiaľ je projekt väčšieho rozsahu, je vhodné použiť niektorý z ďalších nástrojov.

Mocha

`Mocha` [28] je testovací framework, ktorý môže bežať aj v prehliadači, aj v prostredí `NodeJs`. Na rozdiel od `QUnit` je `Mocha` relatívnym nováčikom medzi testovacími nástrojmi JavaScriptu, prvého hlavného vydania sa dočkal v roku 2012.

Pokiaľ je pri projekte využívané automatizovanie testov, ich spúšťanie cez `NodeJS` je jednoznačná voľba. `Mocha` vôbec nevyžaduje k exekúcii testov spúšťať jadro prehliadača, kompletne môžu prebehnúť v `NodeJS`, no ak je asistencia prehliadača žiadúca, je možné testy spúšťať i stade. Je vhodným frameworkom pokiaľ je samotná aplikácia úplne oddelená od testových modulov. Hodí sa skôr na testovanie rozsiahlejších JavaScriptových aplikácii. Neobsahuje vlastnú assertion knižnicu, no existuje mnoho assert nástrojov, ktoré perfektne spolupracujú, najbežnejšou voľbou je spojenie s knižnicou `Chai`. Veľkou výhodou je perfektné zvládnutie testovanie `Ajax` volaní.

Ďalším plusom je odporúčanie na využitie tohto frameworku v oficiálnej dokumentácii k `Redux` architektúre a neposlednom rade pri návrhu testov poskytuje výborne prehľadnú a čitateľnú syntax.

```
describe("My Unit To Test", function () {
  describe("#myMethod()", function () {
    it("should return the thing I expect", function () {
      assert.equal(1, "1", "Passed!");
    });
  });
});
```

Snippet kódu 4.2: Snippet testu napísaného v `Mocha` [28]

Jasmine

`Jasmine` [29] je BDD[7] framework pre JavaScript, ktorý sa stal najpopulárnejším riešením pri testovaní `Angular` aplikácii. `Jasmine` disponuje funkciami určenými na lepšie štruktúrovanie aplikácie a taktiež na `assertion` volania. V mnohých ohľadoch sa podobá frameworku Mocha, no v celkovom ponímaní Mocha poskytuje väčšiu flexibilitu, tým že programátor si môže zvoliť vlastnú Assert knižinicu, ktorú použije. To však len za cenu náročnejšieho nastavenia konfigurácie a následne vzniknutých závislostí. Vhodným riešením je pre `Angular` aplikácie, nakoľko je riadne zdokumentovaný spôsob testovania `Angular` konštrukcii. Na záver ukážka `Jasmine` testu, na ktorom jasne vidieť podobnosť s `Mocha` frameworkom.

```
describe('sorting the list of users', function() {
  it('sorts in descending order by default', function() {
    var users = ['jack', 'igor', 'jeff'];
    var sorted = sortUsers(users);
    expect(sorted).toEqual(['jeff', 'jack', 'igor']);
  });
});
```

Snippet kódu 4.3: Snippet testu napísaného v `Jasmine` [29]

4.3.2. Technológie pre integračné testy

Ako testovaná aplikácia rastie čo do veľkosti, tak i komplexnosti, stáva sa nerealistické spoliehať sa iba na manuálny spôsob verifikácie korektnosti nových súčastí, zachytávanie bugov a kontrolu regresií. Jednotkové testy sú prvou bariérou, ktorú musia chyby prekonať, no niekedy sa stane, že sa chyba prejaví, až pri integrácii komponent, čo sa vymyká pôsobnosti jednotkových testov. Vtedy prichádzajú na rad testy, ktoré sú vykonávané v integračnom prostredí, tzv. `End-to-End` testy [9].

[7] Behavior Driven Developement

Selenium

`Selenium` [30] je súbor nástrojov, ktorých primárnou úlohou je automatizovanie webových aplikácii na testovacie účely, resp. simuláciu reálnych užívateľov. V posledných rokoch stalo najštandardnejším ovládačom prehliadača, pričom komunikácia prebieha skrz `WebDriver`. `WebDriver` je knižnica, ktorá prijíma príkazy od klienta, v tomto prípade od testovacích skriptov a posiela tieto príkazy do prehliadača skrz protokol nazývaný `JSON Wire Protocol`. Výsledkom je vykonanie týchto príkazov v jadre prehliadača a vrátenie výstupom na revíziu. Nevýhodou tohto nástroja je, že na ovládanie je potrebná znalosť Javy, nakoľko testované aplikácie sú naprogramované v JavaScripte.

Protractor

`Protractor` [31] je program na spúšťanie `End-to-End` testov, kompletne napísaný v JavaScripte a bežiaci na prostredí `NodeJS`. Taktiež ako `Selenium`, využíva `WebDriver` na kontrolu jadra prehliadača a simuláciu užívateľských akcii. `Protractor` využíva syntax z `Jasmine` frameworku. Tak ako aj v jednotkovom testovaní, testovací súbor obsahuje jeden alebo viacej `it` blokov, ktoré popisujú požiadavky na aplikáciu, tvorené sú príkazmi `Commands` a očakávaniami `Expectations`. Príkazy vravia Protractoru, aby s aplikáciou vykonal nejakú akciu, napríklad simuloval kliknutie na tlačítko, zatiaľčo očakávania vravia, aký výsledok sa čaká po vykonanej akcii. `Protractor` potom porovnáva očakávania s reálnym výsledkom simulácie.

`Protractor` je primárne určený na E2E[8] testovanie aplikácií vyvinutých vo frameworku `Angular`, no je použiteľný i na testovanie iných aplikácii. `Protractor` je aktívne udržiavaný a vylepšovaný, v blízkej kooperácii s `Angular` tímom.

[8] End-to-End skrátene

NightwatchJs

`Nightwatch.js` [32] je jednoduché riešenie E2E testovania, založené na `NodeJS` prostredí, podobne ako `Protractor`. S nástrojom `Protractor` toho majú mnoho spoločného, tiež využíva mocné rozhranie `Selenium WebDriver` na vykonávanie príkazov. Jeho charakteristickými vlastnosťami je čistá syntax, ktorá umožňuje rýchle a efektívne písanie testov, s využitím Javascriptových či `CSS/Xpath` selektorov. Obsahuje tiež zabudovaný spúšťač testov, ktorý umožňuje spôšťanie testov sekvenčne či paralelne, zgrupovanie testov a podobne. Ďalej má podporu s cloud testovými poskytovateľmi ako `SauceLabs` alebo `BrowserStack` a hlavne existuje skvelá podpora kontinuálnej integrácie so zabudovaných `JUnit XML` reportovaním, čiže je jednoduché integrovanie týchto testov so systémami ako `Teamcity`, `Hudson` či `Jenkins` a to je veľkou výhodou.

```
module.exports = {
  'Demo test Google' : function (browser) {
    browser
      .url('http://www.google.com')
      .waitForElementVisible('body', 1000)
      .setValue('input[type=text]', 'nightwatch')
      .waitForElementVisible('button[name=btnG]', 1000)
      .click('button[name=btnG]')
      .pause(1000)
      .assert.containsText('#main', 'Night Watch')
      .end();
  }
};
```

Snippet kódu 4.4: Ukážka `NightwatchJS` testu [32]

4.3.3. Technológie pre UI testy

Tieto testy majú za úlohu zaručiť, že užívateľské rozhranie pracuje správne. Automatizácia na tejto úrovni je často nákladná na vývoj, i údržbu. Takže dôraz by mal byť kladený

predovšetkým na spodnejšie vrstvy testovej pyramídy 4.5. Pri UI testoch je vhodné sa zameriavať na jednoduché end-to-end workflow a testovať iba súčasti užívateľského rozhrania, ktoré sú naozaj dôležité. Tieto testy bývajú vykonávané až po vykonaní jednotkových testov aj integračných vrámci CI servera, pričom niekedy bývajú od CI servera oddelené úplne a vykonávané sú len v prípade potreby. [21]

Galen

Testovanie layoutu je už z princípu veľmi komplexnou úlohou, ktorú rieši práve tento framework. Testovanie prebieha skrz porovnávanie pozíciovania elementov na stránke, čím poskytuje službu na testovanie responzivity webovej stránky. Využíva špeciálnu popisnú syntax, ktorou sa dá popísať layout akejkoľvek stránky. Tento nástroj môže byť jednoducho integrovaný do seleniového testovania. Jedná sa o open-source framework s kvalitnou dokumentáciou. [33]

4.4. Nástroje na modelovanie procesov

V posledných rokoch bolo vytvorených mnoho softwarových nástrojov špeciálne pre modelovanie a mapovanie procesov. Nástroje popisujú proces a jeho aktivity skrz grafické symboly, štandardom sa stala Business Process Management Notation, popísaná v kapitole 3.7.1. Väčšina týchto nástrojov tiež ponúka analýzy typu ABC[9] alebo tiež simulačné analýzy, čo záleží na prepracovanosti metodológie, ktorú nástroje pre modelovanie procesov podporujú. Tieto nástroje môžu byť rozdelené do troch hlavných skupín [38]:

- *Nástroje pre znázornenie tokov*

  Tieto nástroje sú na najnižšej úrovni a pomáhajú popísať procesy prenesením slovného popisu do grafických symbolov.

- *CASE nástroje*

  Poskytujú konceptuálny rámec pre modelovanie hierarchie procesov a ich popis. Bý-

[9] Activity-Based-Costing

vajú založené na relačných databázach a obsahujú funkcie, ktoré poskytujú možnosti linárnej, statickej a dynamickej analýzy.

- *Simulačné nástroje*

  Poskytujú hlbšiu dynamickú analýzu spojitých alebo deterministických dát, vizualizujú ako objekt putuje konkrétnym procesom. Simulačné nástroje sú väčšinou súčasťou vyspelejších CASE nástrojov.

Nasledujúce odseky budú venované práve analýze dostupných nástrojov.

Progess Savvion

Savvion Process Modeler [34] je jednoducho používateľná business aplikácia na vyjadrenie, simuláciu a dokumentáciu business procesov. Podporuje všetky etapy životného cyklu procesu - návrh, modelovanie, monitorovanie, vizualizáciu i optimalizáciu. Výstupy sú vytvárané na jednotlivých úrovniach (analýza, simulácia, priebeh) na základe definovaných reportov. Progress Savvion je dostupný vrámci skúšobnej 30-dňovej verzie, na ďalšie používanie je potrebná licencia.

IBM Lombardi

Software WebSphere Lombardi [35] od spoločnosti IBM poskytuje flexibilnú platformu na rýchle nasadenie a vylepšenie business procesov. Preslávil sa inovatívnym prístupom k modelovaniu business procesov, čo prilákalo množstvo zákazníkov a neskôr sa tento software stal i významnou akvizíciou spoločnosti IBM. Momentálne poskytuje širokú škálu prostriedkov na modelovanie, monitorovanie, vyhodnocovanie firemných procesov. Založený je na platforme Java EE, čo zaručuje využitie software i mimo operačného systému Windows. Nevýhodami IBM Lombardi je pridrahá licencia a vysoké požiadavky na hardware.

BizAgi Studio

BizAgi Studio[10] [36] taktiež, ako aj predchádzajúca dvojica nástrojov, podporuje všetky etapy životného cyklu firemného procesu. Tiež je založený na platforme Java EE, podporuje implementáciu a vylepšovanie firemných procesov, chýba simulácia. Namodelované procesy je možné exportovať do mnohých formátov, vrátane XPDL - XML Process Definition Language.

Signavio

Signavio [37] je modeler BPM procesov ponúkajúci širokú škálu možností modelovania diagramov. Nástroj funguje ako online služba, čo znamená, že na prácu s ním je potrebné len pripojenie k internetu a webový prehliadač. Hardwarové nároky sú tým pádom oveľa nižšie ako u konkurenčných nástrojov. Podporuje modelovanie v notácii BPMN 2.0, zdieľanie diagramov viacerým užívateľom, čo podporuje tímovú prácu. Signavio má však obmedzené možnosti simulácie, ktorá je možná len skrz externú webovú službu a poskytuje o simulovanom procesi len najzákladnejšie informácie.

[10] skratka od Business Agility

5. Návrh a imlementácia QA procesov

V tejto sekcii bude popísané aké technológie boli do navrhnutých procesov zvolené a prečo, rozbor navrhovaných procesov a ku každému bude pre ilustráciu uvedená i implementácia procesu do konkrétneho prípadu.

V analýze súčasného stavu bolo uvedené, ako funguje spoločnosť InQool, resp. do akého prostredia budú navrhované QA procesy. Okrem toho bol spravený rozbor technológii a nástrojov, ktoré budú pre navrhované procesy potrebné.

5.1. Výber technológii

Výber technológii bude primárne zameraný na vývojový proces projektu Artstaq, sekundárne bude braný ohľad aj na ostatné projekty spoločnosti. Vzhľadom k faktu, že projekt Artstaq je naprogramovaný v technológii `React/Redux`, navrhol som na jednotkové testovanie framework `Mocha` s `Assert` knižnicou `Chai`. Táto technológia je popísaná v sekcii 4.3.1, rozhodujúcim faktorom pri rozhodovaní bolo množstvo dostupnej dokumentácie, čo značne zjednodušuje tvorbu automatických testov. Do projektov založených na JavaScriptovom frameworku `Angular` bude nasadený testovací framework `Jasmine`. V prípade integračných testov bude použitý nástroj na E2E testy `NightwatchJS`, práve pre jednoduchú syntax, kvalitnú dokumentáciu a jednoduchú integráciu do CI serveru Jenkins. Pri ďalšej práci do projektov s technológiou `Angular`, prichádza do úvahy alternatíva nástroja `Protractor`.

Nástroj na modelovanie procesov bude volený na základe použiteľnosti, prehľadnosti, podporovanej notácie a ostatných doplnkových služieb. Zvolený bol nakoniec software Signavio v akademickej verzii, kvôli prepracovanému online riešeniu služieb, vďaka čomu má každý zamestananec s pripojením na internet bezproblémový prístup k diagramom. Diagramy môžu byť exportované do mnohých formátov, čo z neho robí ideálny nástroj na návrh QA procesov.

5.2. Integrácia do CI servera

Mocha a Chai

Konfigurácia spustenia týchto testov bude jednoduchá, nakoľko jednotkové testy sú súčasťou aplikácie. Úlohy, ktoré pracujú s aplikáciou na úrovni NodeJS prostredí sú nadefinované v súbore `package.json` a jedná sa o konkrétne o príkaz na zostavenie aplikácie, ktorý je vykonávaný Jenkinsom. Server je postavený na unixovom systéme, preto príkaz `buildLinux`.

```
{
  name "artstaq-finpartner",
  scripts: {
    start: "node server.js",
    test: "mocha --compilers js:babel-core/register --recursive",
    buildWin: "set \"NODE_ENV=production\" && node build.js",
    buildLinux: "export NODE_ENV=production && node build.js"
  },
  dependencies: {
    ...
  }
}
```

Snippet kódu 5.1: `package.json`

K tomu bude momentálne pridaná ďalšia úloha, na spustenie jednotkových testov. Po nahratí zmeny do Git repozitára bude teda namiesto jedného príkazu teraz vykonaná dvojica príkazov.

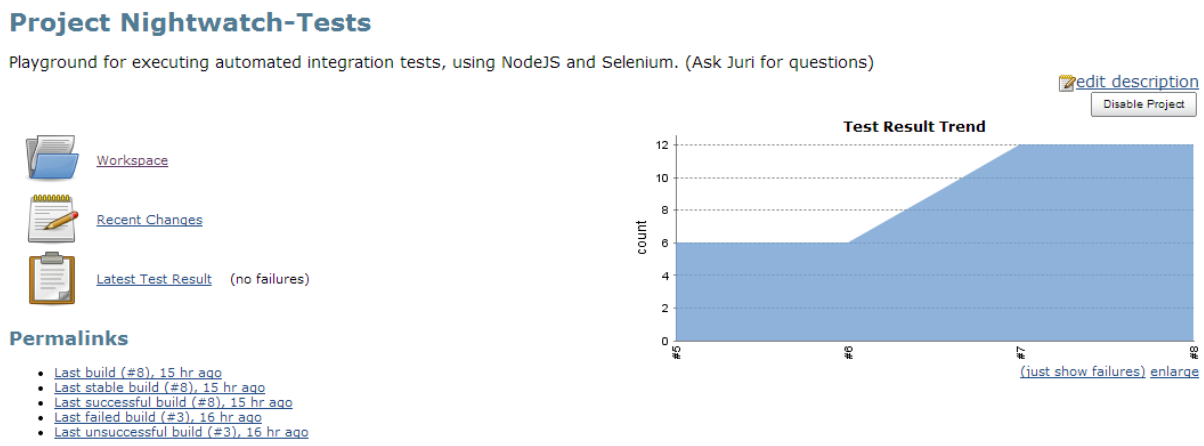
```
npm buildLinux
npm test
```

Nightwatch

Najprv treba zabezpečiť, aby na serveri, kde je nainštalovaný Jenkins boli nainštalované i ďalšie potrebné nástroje ako NodeJS a Mozilla Firefox. Jadro webového prehliadača využíva testovací framework Nightwatch. Ďalším krokom bude nahrať projekt s Nightwatch testami do repozitára v Gitlabe.

```
export NODE_ENV=production && node nightwatch.js --test [tests/fp | tests/ap
| tests/tp]
```

Následne bude jednoduchým príkazom zavolané spustenie testov, zakaždým keď bude zostavená nová verzia aplikácie. Spúšťač testov vygeneruje JUnit kompatibilný XML súbor, ktorý bude ďalej spracovaný systémom Jenkins a výsledky testov budú prehľadne prezentované. Screenshot je zobrazený na obrázku 5.1.



Obr. 5.1: Integrácia NightwatchJS do CI servera Jenkins [25]

5.3. Návrh QA procesov

Firma InQool sa stará vrámci projektu Artstaq o technologickú stránku. Vývojársky tím sa skladá z piatich ľudí - projektového manažéra, dvoch vývojárov na frontend, dvoch vývojárov na backend a tím bude rozšírený vrámci tejto diplomovej práce o testera. Procesy zabezepčovania kvality software budú navrhované v duchu nasledujúcich činností:

- Overenie napĺňania požiadaviek zákazníka, kontrola zhody so zadaním a špecifikáciou projektu
- Kontrola výstupov vývojárov zameraná na dodržiavanie vopred definovaných štandardov a noriem
- Testovanie za účelom hľadania chýb - funkčných, logických, obsahových, systémových

Na overovanie naplnenia zákazníckych požiadaviek budú slúžiť dokumenty so zadanými zákazníckymi scenármi ako aj intenzívna komunikácia so zadavateľom, keďže ten je tiež zapojený do vývoja, ktorý je budovaný v duchu agilných metodík.

Kontrola výstupov a celková organizácia vývoja je podporovaná nástrojom Pivotal Tracker, na túto činnosť bude intenzívne využívaný.

Testovanie bolo na základe *best pratices* zanalyzovaných v teoretickej časti rozdelené medzi programtátorov a testera. Programátori, ktorí pracujú na konkrétnych javascriptových aplikáciách si budú písať vlasné Unit testy. Použitá technológia bude Mocha, pričom samotné testy budú vytvárať mocky[1] z implementovaných komponent. Unit testy budú implementované na báze black box testovania, čiže dôležité je to aby komponenta pri zadaných vstupných parametroch vrátlia po simulovaní správne výstupy. Testy sú však oddelené od samotnej javascriptovej aplikácie (nie úplne, ale budú sa nachádzať v samotnom module), čiže v prípade časovej voľnosti a vrámci rozvíjania svojich testovacích

[1]Pomocné objkety, ktoré simulujú predpokladaný kontext, v ktorom testovaná časť programu pracuje.

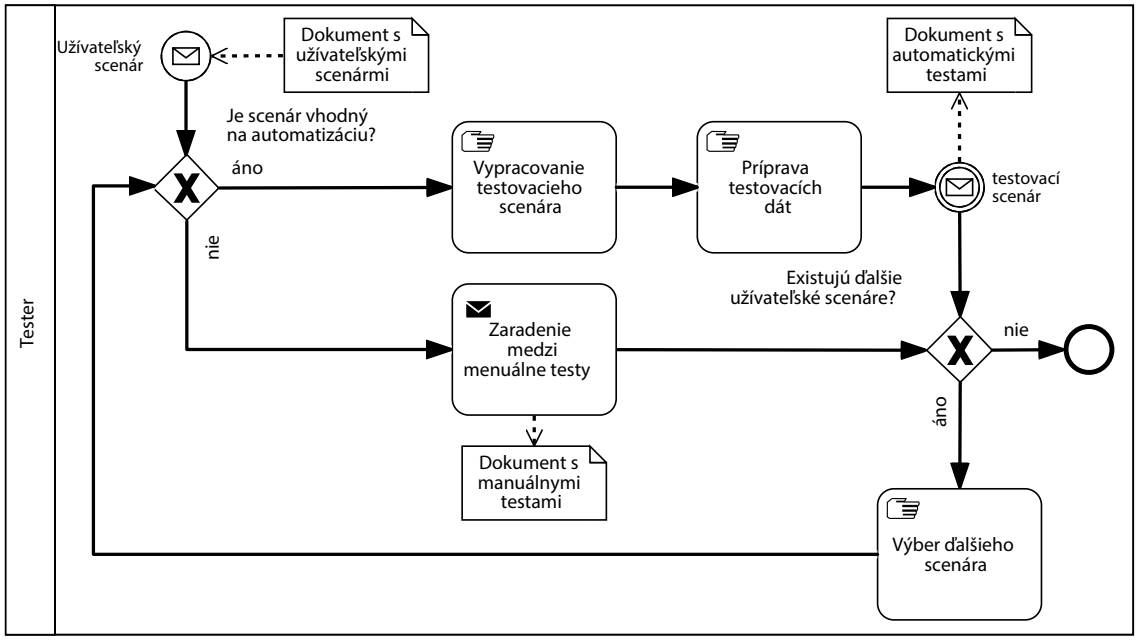
zručností môže vypomáhať i tester. Ďalej tester bude implementovať regresné testy, s dôrazom na automatizáciu. Automatické testy budú spustené vždy pri zostavení novej verzii aplikácie.

5.3.1. QA proces 1 - Tvorba testovacích scenárov

Jendou z hlavných častí pracovnej náplne testera bude navrhovanie testovacích scenárov. Tento proces bude prebiehať medzi *Scrum Mastrom*, čiže projektovým manažérom a testerom.

Návrh Business procesu

Ako vstup do práce testera budú užívateľské scenáre v neformálnom zápise, nakoľko pochádzajú priamo od *Product Ownera*, ktorý nedisponuje dostatočnými programátorskými znalosťami.

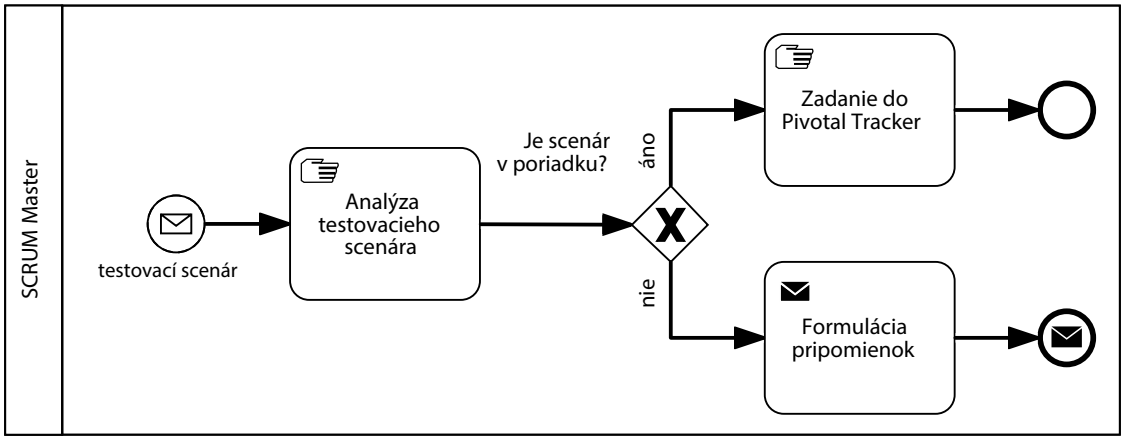


Obr. 5.2: BPM - Tvorba testovacích scenárov (vlastné spracovanie)

Dokument bude mať teda formát zákazníckych požiadavkov zapísaných v bodoch s poku-

som o vytvorenie testovacích scenárov, no bude si to vyžadovať ďalšiu odbornú analýzu. Z dodaného dokumentu bude úlohou testera identifikovať testovacie scenáre a následne ich rozdeliť na tie, ktoré sa budú dať zautomatizovať a tie, ktoré budú musieť byť vykonávané manuálne. Z testovacích scenárov, ktoré sa dajú zautomatizovať bude vybratý práve jeden (ak majú rovnakú prioritu ľubovoľný, inak podľa priotrity) a podľa šablóny, ktorá sa nachádza v prílohe A bude vytovrený scenár automatického testu. Následne budú niektorou technikou z popísaných v kapitole 3.4 predpripravené testovacie dáta. Táto časť procesu je namodelovaná v diagrame 5.2.

Vzniknutý testovací proces bude následne predložený *Scrum Masterovi*, ktorý ho buď odsúhlasí alebo inicializuje úpravy a proces návrhu sa bude opakovať. V prípade, že návrh scenáru automatického testu je v poriadku, bude založený medzi ostatné scenáre a do systému Pivotal Tracer bude testerovi zadaná *User Story* s vypracovaním tohto automatického testu. Tester túto *User Story* prevezme a ohodnotí ju bodmi na základe náročnosti vypracovania. Táto časť procesu je ilustrovaná na diagrame 5.3.



Obr. 5.3: BPM - Tvorba testovacích scenárov (vlastné spracovanie)

Implementácia Business procesu

Vzhľadom k faktu, že projekt Artstaq síce už je prístupný verejnosti, no ešte nie je vo finálnej verzii a jeho údržba bude prebiehať dlhodobo, implementácia automatických testov stále prebieha. Keďže scenárov, či už užívateľkých alebo testovacích bolo vytvorených dosť veľa, bude v tejto kapitole uvedená implementácia jedného z nich, konkrétne - **Prihlasovanie do back office pre finančného partnera**.

Tento užívateľský scenár má nasledovnú formu zápisu v troch bodoch:

1. Môžem sa prihlásiť cez http://artstaq.com aj cez priamu adresu http://fp.artstaq.com
2. Môžem sa odhlásiť
3. V hlavičke vidím prihlasovací email

Na začiatok treba analyzovať užívateľský scenár, či je vhodný na automatizáciu alebo nie. Na automatizáciu sú vhodné vykonávané akcie so systémom, v tomto prípade prihlasovanie i odhlasovanie. Či je v hlavičke zobrazený email užívateľa nepredstavuje žiadnu akciu so systémom, preto tento bod bude spadať do manuálnych testov.

Pri prvom bode sa jedná o otestovanie prihlásenia do aplikácie skrz prihlasovacie formuláre, umiestnené na dvoch rôznych url. Môže vzniknúť niekoľko prípadov, ktoré treba otestovať. Každý z troch typov rolí - finančný partner, art partner, investor - má špecifické povolenia. Partneri majú povolený vstup do svojich administračných rozhraní i obchodnej platformy, investor má prístup iba do obchodnej platformy.
Na otestovanie prihlasovania je najprv nutné preiterovať všetky podstránky, na ktoré je prihlásenie vyžadované. To znamená, že ak pristúpim na takúto podstránku, aplikácia by ma mala automaticky presmerovať na stránku s prihlasovacím formulárom.
Na prípravu testovacích dát do prihlasovacích formulárov je vhodné využiť rozhodovaciu tabuľku, tak ako bola popísaná v teoretickej časti 3.4.1.

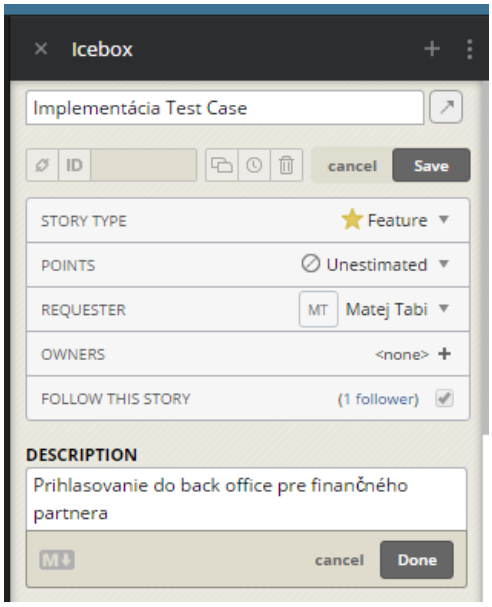
Druhý bod vytvára akciu odhlásenia zo systému, po ktorom opäť nebude možné pristúpiť na podstránky. Otestovať tiež treba situáciu, aby nebolo možné pristúpiť na prihlasovaciu stránku, ak je užívateľ prihlásený. Testovacie dáta pripravovať netreba.

Z uvedenej analázy vzniká testovací scenár do niekoľkých bodov, ktoré sú uvedené v tabuľke 5.1. Vypracovaný testovací scenár podľa šablony v prílohe A je následne uložený do úložiska na Dropboxe medzi ostatné a projektový manažér je o tom notifikovaný. Ten má za úlohu testovací scenár prekontrolovať a pokiaľ je v poriadku, zadá do systému Pivotal Tracker nový tiket, ktorý bude uložený v zložke Icebox.

Krok	**Akcia**	**Očakávaný výstup**
1	Iterovať všetky zamknuté podstránky	Preserovanie na prihlasovaciu stránku
2	Vstup na prihlasovaciu stránku `https://fp.artex500.com/`	Prihlasovacia stránka bude zobrazená
3	Zadané nevalidné užívateľské meno aj heslo	Zobrazená chybová správa
4	Zadané nevalidné meno, validné heslo	Zobrazená chybová správa
5	Zadané validné meno, nevalidné heslo	Zobrazená chybová správa
6	Zadané validné meno aj heslo	Prihlásenie do systému
7	Iterovať všetky zamknuté podstránky	Podstránky budú zobrazené
8	Vstup na prihlasovaciu stránku	Presmerovanie na domovskú užívateľskú stránku
9	Odhlásenie zo systému	Presmerovanie na prihlasovaciu stránku
10	Znova preiterovať všetky zamknuté podstránky	Presmerovanie na prihlasovaciu stránku
11	Kroky 1 - 10 vykonať pre tiež pre `https://artex500.com/`	-

Tabuľka 5.1: Návrh testovacieho scenára

Tento testovací scenár je jeden z mnohých, ktoré boli vrámci nasadzovania QA procesov vytvorené. Do Trade platformy ich bolo vytvorených 46, do Back office finančného partnera 7 a do Back office pre art partnera 11. Týkajú sa systémových akcii ako tvorba Membership kariet, testovanie rôznych formulárov, proces obchodovania atď.



Obr. 5.4: Testovací scenár ako tiket v Pivotal Tracker [42]

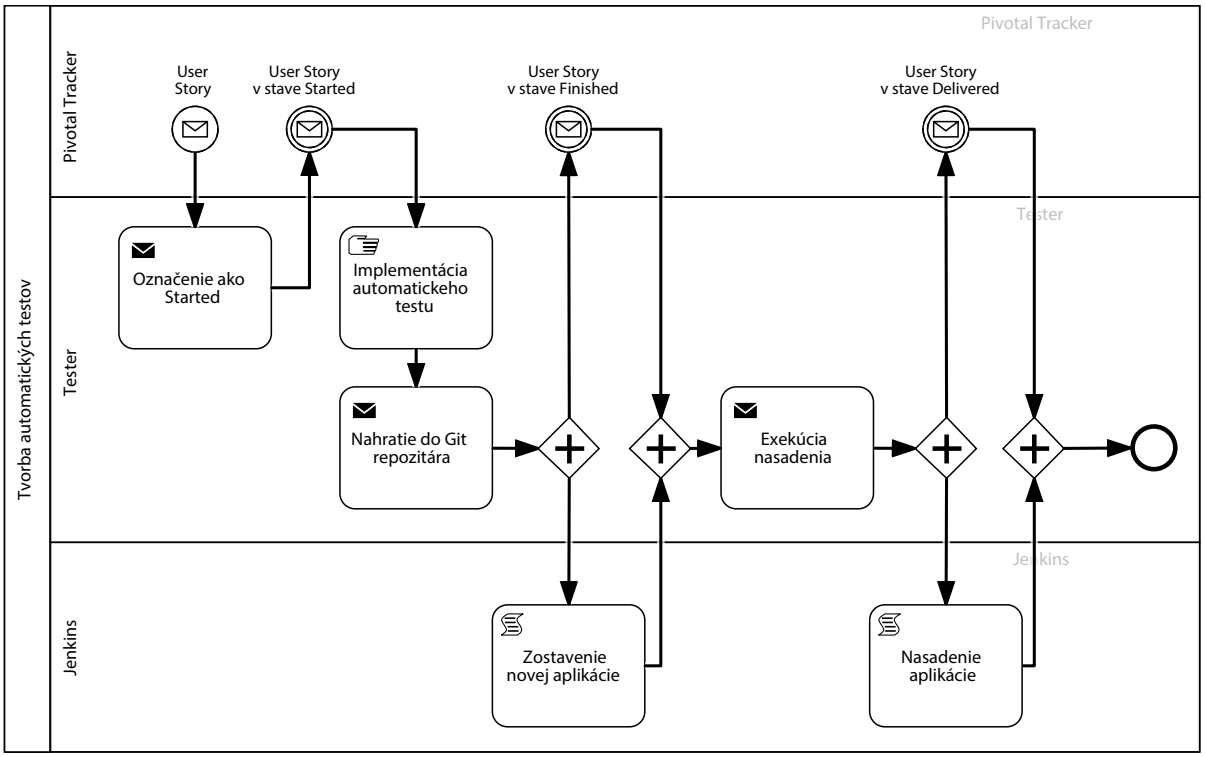
5.3.2. QA proces 2 - Tvorba automatických testov

Automatické regresné testy budú vytvárané za využitia frameworku Selenium a jeho JavaScriptovej nadstavby NightwatchJS, na automatické jednotkové testy bude využitá technológia Mocha + Chai.

Návrh Business procesu

Implementácia automatických testov môže začať po naštudovaní zvolených technológii. V prípade jednotkových testov bude postupované podľa dokumentácie, to znamená, že testami budú pokryté časti zdrojových kódov ako `Action Creators`, asynchrónne `Action Creators`, `Reducery` i komponenty, predstavujúce jednotlivé časti grafického užívateľského rozhrania.

Proces bude prebiehať medzi testerom či vývojárom, v závislosti od toho, kto bude automatické testy implementovať, a *Scrum Masterom*. S tým, že nemalú rolu v tomto procesi zohrajú aj zautomatizované integrované technológie Jenkins a Pivotal Tracker.

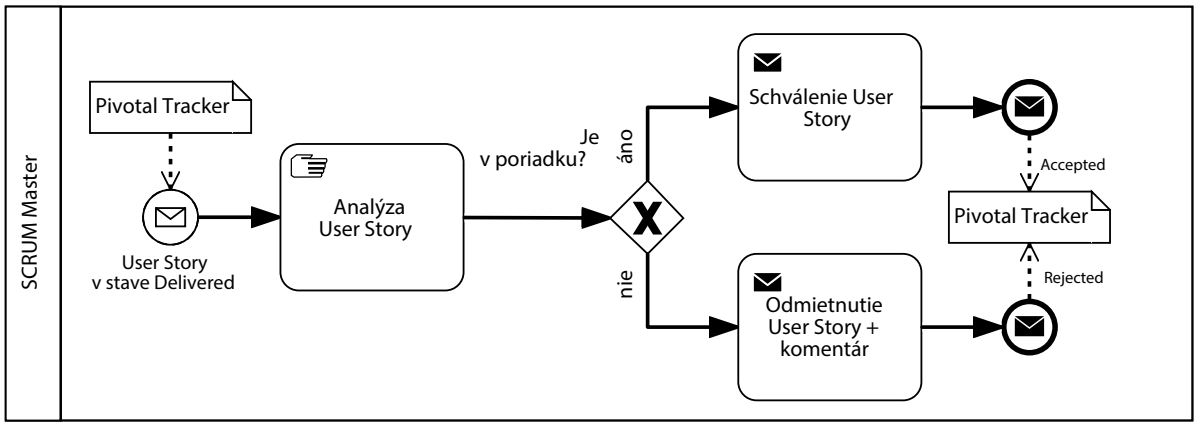


Obr. 5.5: BPM - Tvorba automatických testov (vlastné spracovanie)

Ako vstupné dáta do tohto procesu je zadaný *User Story* z predchádzajúceho procesu. Tento *User Story* sa nachádza zadaný v systéme Pivotal Tracker a zároveň už je obodovaný podľa náročnosti. Tester začne prácu na tomto konkrétnom *User Story* a v systéme Pivotal Tracker ho označí ako *Started*. Následne testy implementuje a dôkladne odskúša. Automatické testy totiž musia byť stopercentne správne, nie je prípustné aby boli chyby v automatických testoch. Ak sú testy hotové, nové verzie zdrojových súborov budú nahrané na server, kde prebehne preklad a výsledkom bude nová verzia aplikácie automatických testov. *User Story* sa v Pivotal Trackeri prepne do stavu *Finished*. Novo preloženú aplikáciu automatických testov bude treba ešte nasadiť spočiatku na integračné prostredie, neskôr i na produkčné. Po nasadení sa *User Story* prepne do stavu *Delivered*.

Po tejto akcii prichádza na rad *Scrum Master*, ktorý odsúhlasí zapracovanú časť, alebo v

prípade, že niečo nie je v poriadku, odmietne ju. Podľa toho je stav *User Story* prepnutý na *Accepted* alebo *Rejected*. Ak je *User Story* odmietnutý, tester bude mesieť zapracovať pripomienky a priebeh sa opakuje od stavu *Started*. Priebeh procesu sa skončí, ak *User Story* bude akceptovaný a v systéme Pivotal Tracker sa bude nachádzať v priečinku *Done*.



Obr. 5.6: BPM - Tvorba automatických testov (vlastné spracovanie)

Implementácia Business procesu

Implementáciu procesu je možné napojiť na implementáciu predchdázajúceho procesu, čiže vstupom do tohto procesu je *User Story* tiket zadaný v Pivotal Trackeri a testovací scenár uložený na Dropboxe.

Tento tiket si prevezme *SCRUM Mastorm* pridelený pracovník, v tomto prípade tester. Ako prvý krok musí ohodnotiť tiket bodmi a to naslednovným spôsobom.

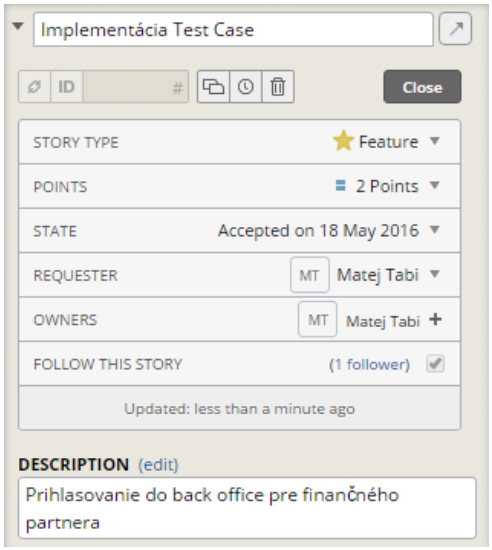
- **0 bodov** - okamžitá úprava
- **1 bodov** - jednoduchá úprava, do pol hodiny
- **2 bodov** - klasická úloha, odhadom na 2 - 3 hodiny
- **3 bodov** - náročnejšia úloha, odhadom na 1 pracovný deň

- **5 bodov** - náročná úloha, potreba rozbiť na menšie tikety

Tento tiket je asi na normálnej úrovni náročnosti, preto mu bude pridelené dva body a následne bude označený ako `Started`. Bude vytvorený samostatný súbor so zdrojovým kódom. NightwatchJS je nástroj, ktorý podporuje spúšťanie i ojedinelých testov, preto je nevyhnutné separovať implementáciu testovacích prípadov do samostatných súborov. Reporty sú generované len na `localhost` úrovni, v CI serveri musia byť dáta štruktúrované, o vizualizáciu sa stará rozšírenie nainštalované do systému Jenkins.

Ak bude test implementovaný a otestovaný, musí byť nahratý do repozitára. V `commit message` musí byť uvedený identifikátor tiketu v Pivotal Trackeri, preto bude mať `commit message` nasledujúci formát:

```
commit -m ‘‘Implemented login page test case [Finished: #2815165]’’
```



Obr. 5.7: Uzaverná implementácia testovacieho scenára [42]

Vďaka vzájomnej integrácii technológii Pivotal Tracer a Gitlab a Jenkins, je po následnom príkaze `push`, User story s identifikátorom #2815165 prehodený do stavu `Finished`

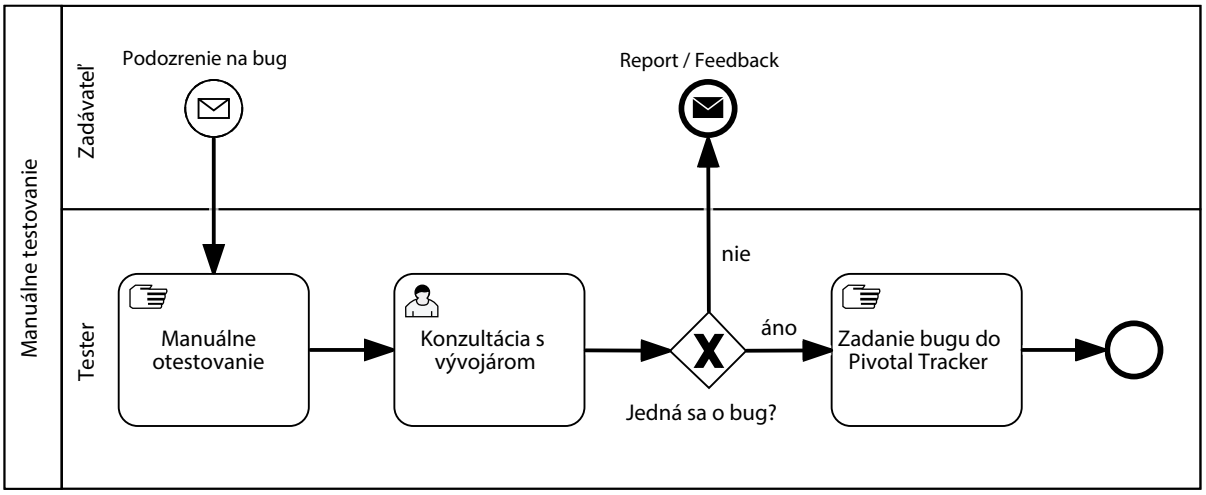
a na Jenkinse je spustené zostavenie novej aplikácie. Po nasadení novej verzie aplikácie na integračné/produkčné prostredie je tiket automaticky prehodný do stavu `Delivered`. Potom k tiketu prijde zadávateľ, čiže produktový manažér a tiket odsúhlasí. Následne bude presunutý do sekcie `Done`. Týmto proces novej `feature` test aplikácie končí a môže začať implementácia ďalšieho testovacieho scenára či iná aktivita.

5.3.3. QA proces 3 - Manuálne testovanie

Automatické testy síce uľahčujú prácu testerovi, no nie vždy je vhodné využívať túto možnosť. Niektoré veci treba otestovať manuálne.

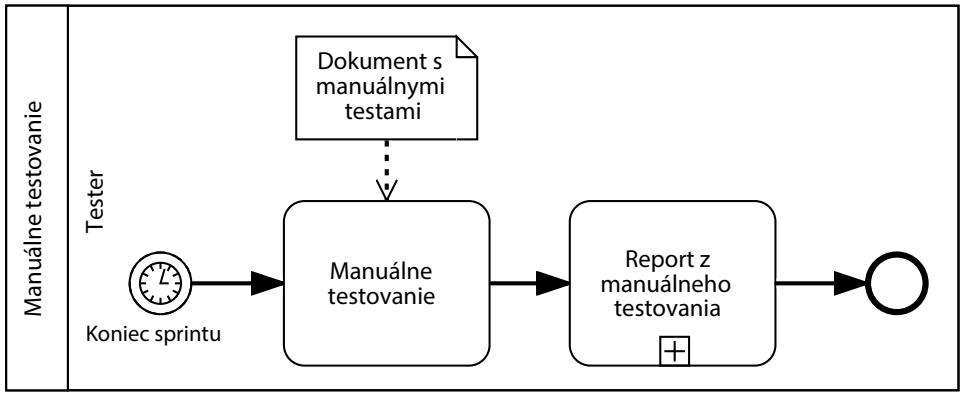
Návrh Business procesu

Konkrétne ak príde požiadavok od nadriadeného na otestovanie súčasti systému s podozrením na bug, je vhodnejšie to skontrolovať manuálne a vzápätí priamo reportovať. Na komunikáciu môže byť využitý systém Pivotal Tracker, prípadne jeden z rýchlejších komunikačných kanálov, napríklad Slack, ktorý je vo firme InQool zaužívaný, prípadne osobná konzultácia s vývojárom v duchu agility.



Obr. 5.8: BPM - Manuálne testovanie (vlastné spracovanie)

Tiež sa môže v User Stories vyskytnúť užívateľský scenár, ktorý nebude vhodné zautomatizovať. Napríklad pri implementácii procesu v sekcii 5.3.1 sa v zadaní užívateľského scenára nachádza bod - **V hlavičke vidím prihlasovací email**, tento bod nepredstavuje nijakú akciu so systémom. Teoreticky by sa dal tento bod automatizovať nástrojom Galen, ktorý sa využíva na testovanie responzivity webov, no Artstaq nie je stavaný responzívne, na zariadeniach s menším rozlíšením je stanovená minimálna šírka 1000 pixelov. Responzivitu preto testovať nie je potrebné a tým pádom tento bod spadá do manuálnych testov. Zo zadaných užívateľských scenárov preto budú vyseparované body, ktoré sa budú testovať manuálne a na konci každého sprintu budú musieť byť manuálne otestované všetky, ktorých sa nejakým spôsobom zmeny dotkli.



Obr. 5.9: BPM - Manuálne testovanie (vlastné spracovanie)

Do kategórie manuálnych testov tiež patrí i kontrola nových súčastí informačného systému, ktoré nie sú dostatočne triviálne na schválenie *SCRUM Masterom*, pričom zodpovednosť preberie tester. Implementácia procesu manuálneho testovania bude zobrazená práve na tomto prípade. Proces je identický s 5.6 s jediným rozdielom, že ho vykonáva iná osoba - tester.

Implementácia Business procesu

Vstupným *User Story* je **Google Analytics - nastavenie ecommerce a udalostí**, čo znamená aké udalosti má sledovať Google Analytics, kvôli strategickým zámerom, ako napríklad aby sa vedelo, ktoré diela obchodnej platformy sú najviac prezerané, u ktorých udalostí je najvačší odlyv zákazníkov a podobne.

Formát *User Story* je 20 bodov s udalosťami po ktorých sa majú sledovať dané štatistiky.

1. Kliknutie na dielo:
   `ga('send', 'event', 'Artworks', 'Show detail', '[artwork name]');`
2. Kliknutie na umelca:
   `ga('send', 'event', 'Artists', 'Show detail', '[artist name]');`
3. Kliknutie na Trade Button kdekoľvek:
   `ga('send', 'event', 'Trade', 'Trade - begin', '[artwork name]');`
4. Neúspešná platba:
   `ga('send', 'event', 'Trade', 'Trade - Unsuccessful', '[artwork name]');`
5. Úspešné nastavenie Delviery Address:
   `ga('send', 'event', 'DeliveryAddress', 'Set');`
6. ...

Vývojár implementuje danú funkcionalitu do informačného systému, nasadí na testovacie prostredie a tým sa *User Story* nachádza v stave *Delivered*. Následne tento tiket prevezme nadriadený, čiže *Scrum Master* alebo priamo *Product Owner* a vzhľadom k tomu, že následné schválenie je podmienené vyskúšaním každej akcie a kontroly udalosti v Google Analytics, predá zodpovednosť za kontrolu tohto tiketu testerovi.

Kontrola tohto tiketu si vyžaduje podrobnú znalosť systému Artstaq, keďže mnohé udalosti môžu byť vyvolané viacerými spôsobmi, tester musí manuálne prekontrolovať všetky

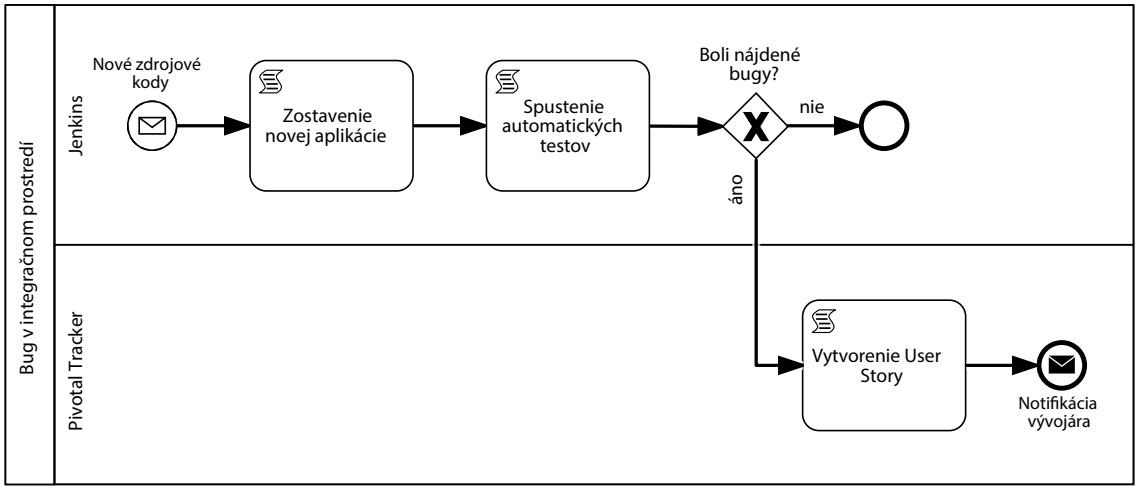
možnosti. Pri kontrole konkrétne tohto tiketu boli nájdené 3 bugy, takže po kontrole tiketu testerom sa tiket nachádzal v stave *Rejected*.

5.3.4. QA proces 4 - Monitorovanie a reportovanie

Automatizácia pri tomto procesi zohráva hlavnú rolu, keďže monitoring počas integrácie i produkcie bude prebiehať automaticky a ak sa vyskytnú nejaké bugy, príslušný vývojár bude notifikovaný.

Návrh Business procesu

Reporting bude prebiehať ako proces zobrazený na diagrame 5.10. Proces bude tentokrát v réžii vývojára a testera a vďaka integrácii systémov Jenkins a Pivotal Tracker bude do značnej miery zautomatizovaný.

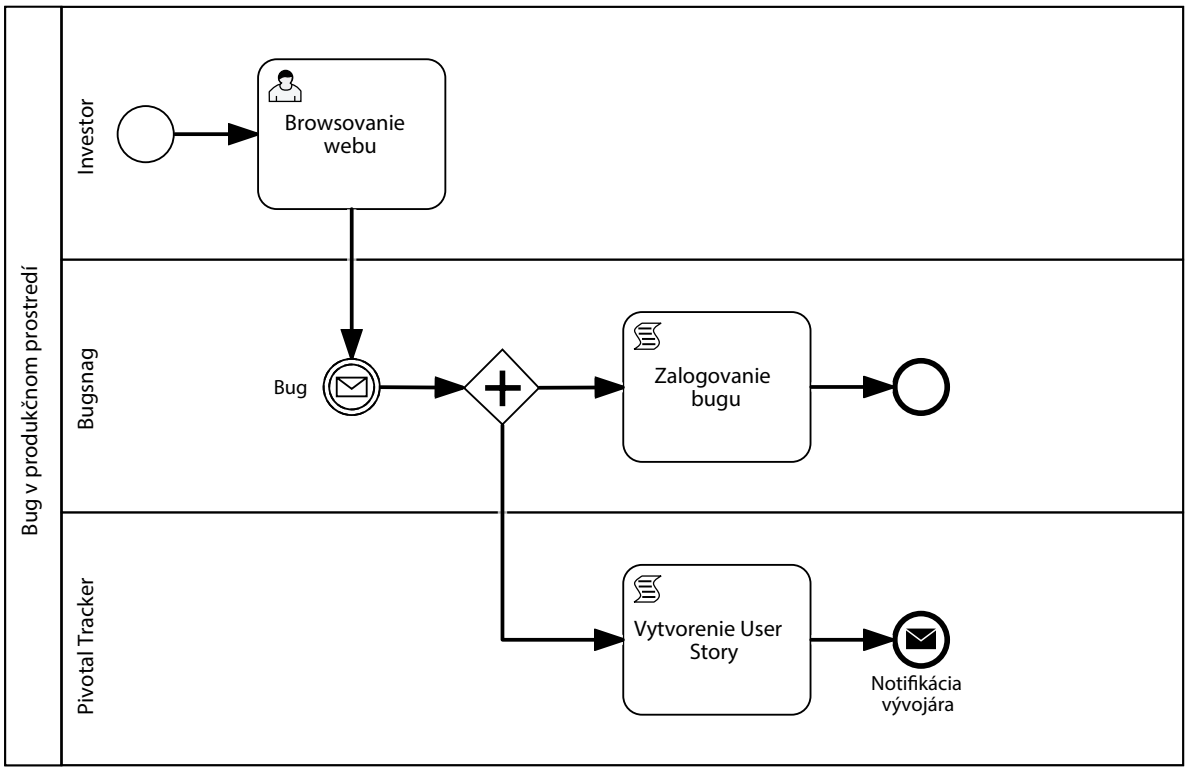


Obr. 5.10: BPM - Monitorovanie a reportovanie (vlastné spracovanie)

Testy budú na Jenkins-e spustené automaticky po každom zostavení novej verzie javascriptovej aplikácie frontendu. To znamená, že ak niektorý z testov skončí neúspechom, bude automaticky zaslaná spáva do Pivotal Trackeru a *User Story* bude priradený testerovi.

Tester najskôr zkonzultuje bug s vývojárom, ktorý spravuje danú časť aplikácie a podľa toho, či sa jedná o triviálnu záležitosť alebo nie bude nasledovať ďalší postup. Buď bude tiket vybavený okamžite, alebo ak sa jedná o závažnejší bug, bude nasledovať konzulácia vrámci Pivotal Trackeru a klasický životný cyklus *User Story* od stavu Started až po Accepted.

Posledná časť navrhovaného QA je v podstate monitoring produkčne fungujúcej aplikácie. Jeho zmyslom je zachytiť bug, ktorý nastane na strane užívateľa resp. investora.



Obr. 5.11: BPM - Monitorovanie a reportovanie (vlastné spracovanie)

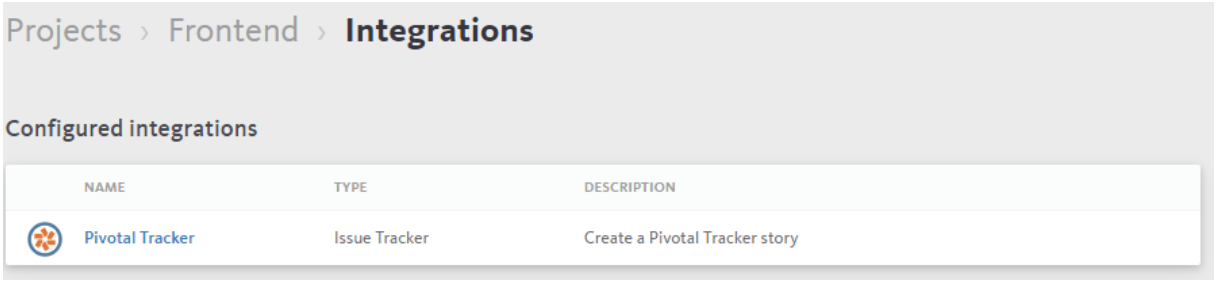
Nástroj, ktorý bude na zachytávanie bugov využitý, je Bugsnag. Poskytuje služby na monitoring zdrojových kódov v programovacom jazyku Java i JavaScript, takže je možné jednu licenciu využiť na monitoring frontendu, tak i backendu. V prípade, že v produkčnom prostredí nastane bug, skrz Bugsnag je ihneď tento bug zalogovaný a integrácia so

systémom Pivotal Tracker zabezpečí, že príslušnému vývojárovi - ktorý naposledy upravoval zdrojové kódy, ktoré spôsobili bug - bude pridelený *User Story* typu *Bug*.

Implementácia Business procesu

Implementácia automatického reportingu spočíva v inštalácii **Notification Plugin** do CI serveru. Plugin musí byť ďalej nakonfigurovaný tak, aby zasielal správu vo formáte JSON do dashboardu v Pivotal Trackeri, ktorý je identifikovaný unikátnym id a tokenom.

Reporting taktiež spočíva v konfigurácii nástroja, tentokrát Bugsnag, ktorý poskytuje možnosť integrácie s Piovtal Trackerom priamo v užívateľských nastaveniach.



Obr. 5.12: Bugsnag [43]

6. Vyhodnotenie

QA manažér sa stretáva s nátlakom dvoch hlavných požiadavkov, ktoré sú zároveň účelom tejto diplomovej práce:

- testovať rýchlejšie
- dodávať software s menej defektami

Zlepšenie týchto dvoch faktorov vyžaduje vyvážený mix ľudí, procesov a nástrojov. Ten bol navrhnutý vrámci predošlej kapitoly, je taktiež dôležité stanoviť si podporné KPI[1] - kľúčové ukazatele výkonnosti, ktoré základné dva ciele dopomôžu dosiahnuť. Inými slovami, zvýšenie kvality produkovaného software a skĺbenie s monitorovaním KPI metrík je kľučom k využitiu potenciálu Quality Assurance naplno. [39]

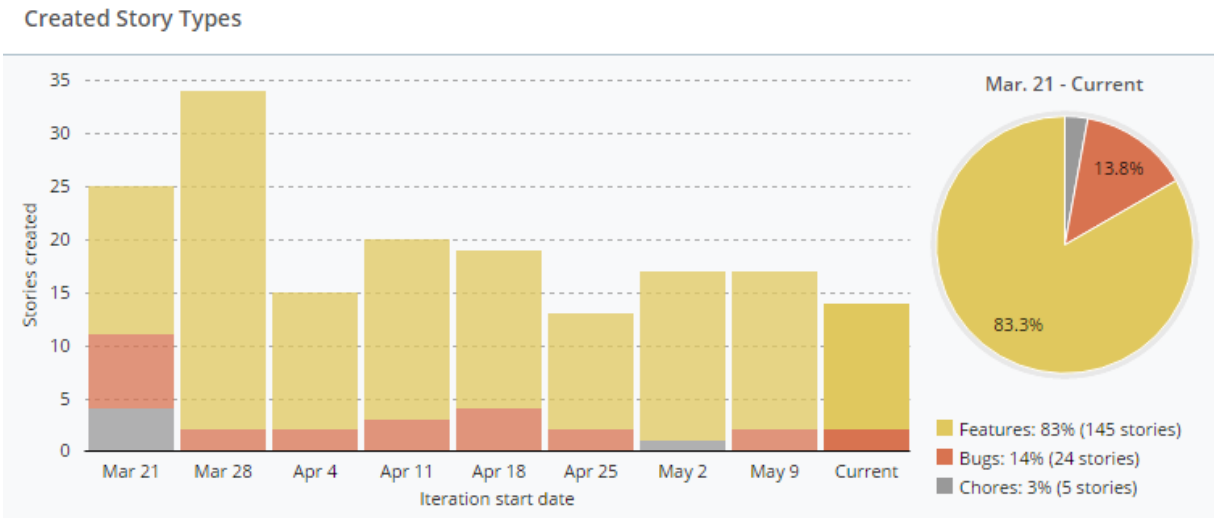
6.1. KPI metriky

K navrhnutým QA procesom bolo preto navrhnutých a sledovaných týchto 8 KPI metrík.

1. **Active Defects** - Aktívne defekty

   Trackovanie aktívnych bugov je jednoduchou KPI metrikou, monitorovateľnou okamžite. Lepšie je, keď sa hodnoty tejto metriky pohybujú v nižších číslach. Aktívny defekt znamená, že bol vytvorené vrámci manažérskeho systému, zapracovaný, či čaká na schválenie. Jednoducho, ak sa na defekte začne akokoľvek pracovať, spadá do tejto metriky. Stanovenie hranice, koľko aktívnych defektov je pri vývoji projektu záleží od fázy životného cyklu, v ktorej sa projekt nachádza, tak i od komplexnosti projektu. Táto metrika bude projekt Artstaq stanovená na 15% z celkového počtu zapracovávaných súčastí software. Na nasledujúcom grafe 6.1 je zobrazený počet tiketov, ktoré boli vytvorené vrámci systému Pivotal Tracker a podiel bugov na celkovej práci počas jednotlivých sprintov. Ak presiahne podiel bugov počas tejto fázy vývoja projektu, bude treba podniknúť opatrenia.

[1] Key Performance Indicators



Obr. 6.1: Active Defects [42]

2. **Authored tests** - Schválené testy

   Táto KPI metrika je dôležitá, nakoľko udáva počet schválených testovacích scenárov, čiže monitoruje predovšetkým návrhársku aktivitu testera. S pribúdajúcimi požiadavkami je potreba rozhodovať, ktoré z užívateľských scenárov je vhodné zautomatizovať do regresných testov a ktoré budú zaradené medzi manuálne testy, vykonávané ad hoc v prípade potreby. Merať sa bude pomer Authored Tests s celkovým počtom užívateľských scenárov. Hodnota tejto KPI metriky bude stanovená na 100%, čo znamená , že ku každému User Story bude vytvorený zároveň testovací scenár, buď manuálny či automatický.

3. **Covered Requirements** - Pokrytie požiadavkov

   Týmto KPI sa meria percentuálne pokrytie zákazníckych požiadavkov aspoň jedným testom. Cieľom je dosiahnutie hodnoty 100%, čo znamená, že každý každý požiadavok bude podchytený nejakým, či už automatickým či manuálnym testom.

4. **Automated tests** - Automatizácia testov

   Rozhodovanie čo testovať automaticky a čo netestovať automaticky, ide ruka v ruke spolu s rozpočtom na testovacie účely software. Vo všeobecnosti platí, že čím viacej testov je automatizovaných, tým je väčšia pravdepodobnosť, že budú zachytené

bugy ešte pred tým ako bude aplikácia nasadená na produkčné prostredie. Metrika bude spočiatku stanovená na hranici 20% a postupne ako sa bude QA tím vyvíjať a nadobúdať skúsenosti, latka sa bude posúvať nahor.

	Trade Platforma	Fin Partner Backoffice	Art Partner Backoffice
Počet užívateľských scenárov	104	19	63
Počet zautomatizovaných scenárov	46	7	11
Percentuálne vyjadrenie	**44%**	**37%**	**17%**

Tabuľka 6.1: Počet scenárov

5. **Passed Tests** - Úspešné testy

   Táto KPI metrika je jedna z tých, ktoré môžu byť klamlivé. To, že táto metrika má vysoké percentá úspešnosti, nemusí automaticky znamenať, že software je bez chýb. Práve naopak, ak je vysoké percento úspešných testov a bugy sa objavujú až pri produkčnej verzii software, niečo nie je v poriadku.

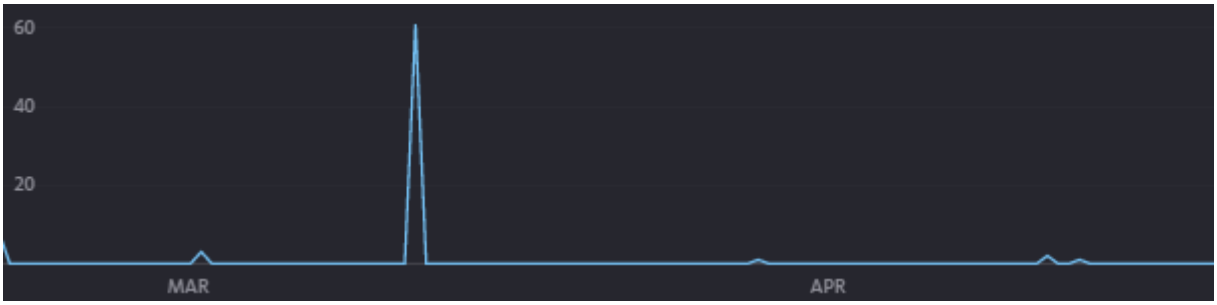
6. **Rejected Defects** - Zamietnuté defekty

   Sú to defekty nájdené testrom v produkte, ale vývojár ich neakceptoval. Ak vývojový tím zamieta vysoké percento defektov so slovami *pracuje tak ako je v zadaní*, je vhodná doba prejsť si projektovú dokumentáciu spolu s celým tímom, aby nedochádzalo k nedorozumeniam. Percentuálne by nemalo byť odmietnutých viac ako 5% reportovaných bugov.

7. **Severe Defects** - Závažné defekty

   Jedná sa o výbornú KPI metriku, no treba si presne stanoviť, ktoré defekty sa budú považovať za závažné. Vrmáci projektu Artstaq bola táto metrika stanovaná na bugy, ktoré prijdú z produkčného prostredia a ich počet bude obmedzený na maximálne 5 za mesiac. Snahou je samozrejme minimalizácia tohto počtu. Na diagrame zo služby Bugsnag 6.2, je zobrazený počet závažných porúch užívateľského rozhrania. Vidieť, že v polovici aplíla sa do produkcie dostala chyba ktoré spôsobovala mnoho pádov systému. Takéto kritické chyby sa musia odhaľovať už počas integrá-

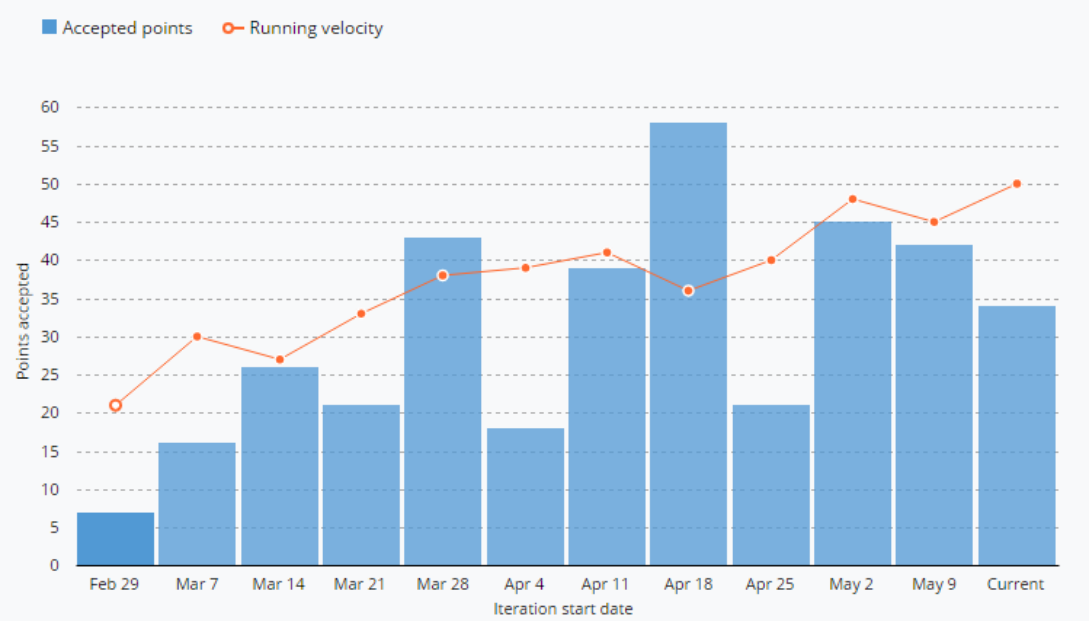
cie systému.



Obr. 6.2: Závažné defekty [43]

8. **Tests Executed** - Počet vykonaných testov

   Táto metrika zahŕňa všetky vykonané testy, manuálne i automatické. Ide o tzv. deskriptívnu metriku, ktorá dopomáha sledovať spád testovacieho procesu vrámci jednotlivých sprintov. Graf 6.3 je vyexportovaný z Pivotal Trackeru a zobrazuje súčet bodov testov, ktoré boli vrámci konkrétnych sprintov akceptované.



Obr. 6.3: Vykonané testy [42]

6.2. Zhrnutie

Vrámci prakticej časti boli implementované QA procesy zamerané na frontend informačných systémov. Tieto procesy, spolu so sledovanými metrikami tvoria počiatočné nastavenie Quality Assurance v spoločnosti InQool, vzhľadom k tomu, že v predchádzajúcom fungovaní QA absentovalo. Implementáciou procesov boli jednoznačne určenené role členov tímu, ktoré nesú za zabezpečovanie kvality zodpovednosť a tiež boli určené metriky, na základe ktorých sa bude dať s navrhnutými procesmi ďalej pracovať a optimalizovať ich. Programátori boli odbremenení od vyčerpávajúceho manuálneho testovania po každom nasadení. A automatické regresné testy dávajú po každom nasadení aplikácie nálepku istoty, že všetky súčasti po úpravách fungujú podľa očakávaní.

V ďalšej práci, ktorá by mohla byť zameraná na optimalizáciu týchto procesov sa naskytujú ďalšie ciele, ktoré by mohli optimalizovať tieto procesy a pedantnejšie vyčísliť dopad na fungovanie spoločnosti.

7. Záver

Vrámci tejto diplomovej práce boli do spoločnosti InQool a.s. navrhnuté a implementované procesy, ktoré zvyšujú kvalitu vyvíjaných produktov. Navrhnuté boli konkrétne 4 Quality Assurance procesy, implementované do projektu Artstaq, v ktorom spoločnosť InQool zastáva technologickú stránku. QA procesy predtým v spoločnosti neboli vôbec nastavené, preto je výsledkom tejto práce inicializačné nastavenie fungujúcich procesov, ktoré sa budú v ďalšom fungovaní spoločnosti ďalej zlepšovať a optimalizovať.

Procesy boli navrhnuté tak, aby zapadli do firemnej kultúry, ktorá je založená na agilných metodikách a priamo sa orientuje na rozvoj človeka a jeho profesných i osobnostých čŕt. Podctivo boli vyraté moderné technológie testovania software, v spoločnosti bola vytvorená pracovná pozícia, ktorá dovtedy nemala v tíme spoločnosti miesto - tester. Navrhované QA procesy spadajú predovšetkým do jeho kompetencie - návrh testovacích scenárov, testovacích dát, implementácia automatických testov, vykonávanie manuálnych testov.

Nakoniec boli nastavené KPI metriky, ktoré sú sledované a zaznamenávané, aby bolo možné v budúcnosti sledovať vývoj, ktorým sa stav projektov uberá a v neposlednom rade, aby bolo možné v budúcnosti QA procesy ďalej zoptimalizovať na základe predchádzajúcich meraní a v prípade nepriaznivého vývoja podniknúť protiopatrenia.

Hlavný prínos diplomovej práce pozostáva z prvotnej analýzy a návrhu QA procesov, nielen pre softwarovú spoločnosť InQool a.s., ale na základe týchto vytvorených aktivít je ich možné využiť i v ďalších spoločnostiach so zameraním na vývoj IT projektov. S nimi je spojený aj návrh indikátorov pre monitorovanie kvality vyvíjaného software. Navrhované QA procesy boli do spoločnosti implementované a ich implementácia stále prebieha. Víziou do budúcnosti a ďalší rozvoj myšlienky, ktorú nesie táto práca, je implementácia procesov i do ostatných projektov, na ktorých sa spoločnosť InQool podieľa.

Literatúra

[1] Puolitaival, O.P. 2008. *Adapting model-based testing to agile context*. Edita Prima Oy, Helsinki.

[2] Bruckner, T. (2012). *Tvorba informačních systému*. Praha: Grada.

[3] Ana C. R. Paiva, PhD Thesis. *Automated Model-Based Testing of Graphical User Interfaces*.

[4] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa: A Planning-Based Approach to GUI Testing. *In Proceedings of The 13th International Software/Internet Quality Week*, May 2000.

[5] Mariani, L., Pezze, M., Riganelli, O. and Santoro, M. (2014). *Automatic testing of GUI-based applications. Software Testing, Verification and Reliability*, 24(5), pp.341-366.

[6] Lewis, W. and Veerapillai, G. (2005). *Software testing and continuous quality improvement*. Boca Raton: Auerbach Publications.

[7] Schulmeyer, G. (2008). *Handbook of software quality assurance*. Boston: Artech House.

[8] Singh, Y. (2011). *Software testing*. Cambridge: Cambridge University Press.

[9] Myers, G., Sandler, C. and Badgett, T. (2012). *The art of software testing*. Hoboken, N.J.: John Wiley & Sons.

[10] Anon, (2016). [online] Dostupné z: http://martinfowler.com/articles/continuousIntegration.html [cit. 22 May 2016].

[11] Layton, M. (n.d.). *Scrum for dummies*.

[12] Ashmore, S. and Runyan, K. (2014). *Introduction to agile methods*. Harlow: Addison--Wesley.

[13] Řepa, V. (2012). *Procesně řízená organizace*. Praha: Grada.

[14] Přibek, J. (2004). *Systémy managementu jakosti*. Praha: Národní informační středisko pro podporu jakosti.

[15] Drahotský, I. and Řezníček, B. (2003). *Logistika*. Brno: Computer Press.

[16] Grasseová, M., Mašlej, M. and Brechta, B. (2010). *Manažerské rozhodování*. Brno: Univerzita obrany.

[17] Šmída, F. (2007). *Zavádění a rozvoj procesního řízení ve firmě*. Praha: Grada.

[18] Burlton, R. (2001). *Business process management*. Indianapolis, Ind.: Sams.

[19] Weske, M. (2007). *Business process management*. Berlin: Springer.

[20] Mendling, J., Weidlich, M. and Weske, M. (2010). *Business process modeling notation*. Berlin: Springer.

[21] TDD, J. (2016). *JavaScript unit test tools for TDD*. [online] Stackoverflow.com. Dostupné z: http://stackoverflow.com/questions/300855/javascript-unit-test-tools-for-tdd [cit. 22 May 2016].

[22] Facebook.github.io. (2016). *A JavaScript library for building user interfaces — React*. [online] Dostupné z: https://facebook.github.io/react/ [cit. 22 May 2016].

[23] Redux.js.org. (2016). *Read Me — Redux*. [online] Dostupné z: http://redux.js.org/ [cit. 22 May 2016].

[24] Angularjs.org. (2016). *AngularJS — Superheroic JavaScript MVW Framework*. [online] Dostupné z: https://angularjs.org/ [cit. 22 May 2016].

[25] Jenkins.io. (2016). *Jenkins*. [online] Dostupné z: https://jenkins.io/ [cit. 22 May 2016].

[26] Google Testing Blog. (2016). *Just Say No to More End-to-End Tests*. [online] Dostupné z: http://googletesting.blogspot.cz/2015/04/just-say-no-to-more-end-to-end-tests.html [cit. 22 May 2016].

[27] jquery.org, j. (2016). *QUnit*. [online] Qunitjs.com. Dostupné z: https://qunitjs.com/ [cit. 22 May 2016].

[28] Mochajs.org. (2016). *Mocha - the fun, simple, flexible JavaScript test framework*. [online] Dostupné z: https://mochajs.org/ [cit. 22 May 2016].

[29] Jasmine.github.io. (2016). *introduction.js*. [online] Dostupné z: http://jasmine.github.io/edge/introduction.html [cit. 22 May 2016].

[30] Seleniumhq.org. (2016). *Selenium - Web Browser Automation*. [online] Dostupné z: http://www.seleniumhq.org/ [cit. 22 May 2016].

[31] Protractortest.org. (2016). *Protractor - end to end testing for AngularJS*. [online] Dostupné z: http://www.protractortest.org/#/ [cit. 22 May 2016].

[32] Nightwatchjs.org. (2016). *Nightwatch.js*. [online] Dostupné z: http://nightwatchjs.org/ [cit. 22 May 2016].

[33] Galenframework.com. (2016). *Galen Framework — Automated testing of responsive design*. [online] Dostupné z: http://galenframework.com/ [cit. 22 May 2016].

[34] Progress.com. (2016). *Business process management*. [online] Dostupné z: https://www.progress.com/openedge/components/bpm[cit.22May2016].

[35] Www-01.ibm.com. (2016). *IBM - WebSphere Lombardi Edition - Software*. [online] Dostupné z: https://www-01.ibm.com/software/integration/lombardi-edition/ [cit. 22 May 2016].

[36] cez, V. (2016). *Process automation and workflow software*. [online] Bizagi.com. Dostupné z: http://www.bizagi.com/en/products/bpm-suite/studio [cit. 22 May 2016].

[37] Signavio.com. (2016). *BPM, process modeling and BPMN 2.0 with Signavio — Collaborative Process Design*. [online] Dostupné z: http://www.signavio.com/[cit. 22May2016].

[38] Fischer, L. (2006). *2007 BPM and workflow handbook*. Lighthouse Point, Florida: Future Strategies, Inc.

[39] ResultsPositive. (2015). *12 Key Performance Indicators for QA & Test Managers*. [online] Dostupné z: http://resultspositive.com/12-key-performance-indicators-for-qa-test-managers/ [cit. 22 May 2016].

[40] Softwaretestingmentor.com. (2016). *Test Design Techniques*. [online] Dostupné z: http://www.softwaretestingmentor.com/test-design-techniques/ [cit. 22 May 2016].

[41] Artex500.com. (2016). [online] Dostupné z: https://artex500.com/[cit. 23May2016].

[42] Pivotaltracker.com. (2016). *Pivotal Tracker*. [online] Dostupné z: https://www.pivotaltracker.com/dashboard [cit. 23 May 2016].

[43] Bugsnag. (2016). Bugsnag. [online] Dostupné z: https://bugsnag.com/ [cit. 23 May 2016].

8. Prílohy

A. Šablóna na Test Case

Test Case #: 1.1 **System:** Artstaq

Test Case Name: **Short description**:

Pre-conditions:

- Item1
- Item2
- ...

Step	Action	Expected system response	Pass/Fail	Comment
1				
2				
3				
4				
5				
6				
7				
8				

Post-conditions:

1. Item1
2. Item2
3. ...