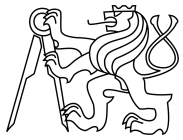
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Kvalitativní parametry webových aplikací na platformě IBM BPM
Student:	Bc. Ivan Prokipčák
Vedoucí:	Ing. Pavel Náplava
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Pro vývoj aplikací jsou často používány vývojové "frameworky", které zrychlují vývoj. Jejich nesprávné užití způsobuje, že výsledná aplikace je na jedné straně obtížně použitelná uživateli a na druhé straně, z pohledu dodavatele, obtížně spravovatelná a rozšiřitelná. V rámci diplomové práce navrhněte a ověřte metodiku, která tyto problémy minimalizuje. Postupujte následovně:
- Proveďte rešerši parametrů kvality webových aplikací a identifikujte klíčové parametry provozu s vysokým počtem uživatelů, včetně metod jejich hodnocení.
- Proveďte rešerši metod vývoje aplikací na platformě IBM BPM, včetně vlivu na kvalitu výsledné aplikace.
- Na základě výstupů rešerší analyzujte aplikace "Závěrečné práce" a "Doktorské studium", provozované ČVUT FIT.
- Vyhodnocením dříve nalezených parametrů identifikujte příčiny problémů aplikací a navrhněte jejich řešení formou metodiky vývoje.
- Metodiku aplikujte na analyzované aplikace, výsledky ověřte v laboratorních podmínkách a vyhodnoťte její přínosy.

Seznam odborné literatury

- Deliver Modern UI for IBM BPM with the Coach Framework and Other Approaches (https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248355.html?Open)
- Business Process Management Design Guide: Using IBM Business Process Manager (https://www.redbooks.ibm.com/abstracts/sg248282.html?Open)
- L. Olsina, G. Covella, G. Rossi, Web Engineering, chapter Web Quality, Springer, ISBN: 978-3-540-28196-2
- L. Olsina, G. Rossi, A. Garrido, D. Distante, G. Canfora, Incremental Quality Improvement in Web Applications Using Web Model Refactoring, Springer, ISBN: 978-3-540-77009-1
- M. Herrera, M. a. Moraga, Quality in Use Model for Web Portals (QiUWeP), Springer, ISBN: 978-3-642-16984-7
- P. Lew, L. Olsina, L. Zhang, Quality, Quality in Use, Actual Usability and User Experience as Key Drivers for Web Application Evaluation, Springer, ISBN: 978-3-642-13910-9

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 15. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práca

Kvalitativní parametry webových aplikací na platformě IBM BPM

Bc. Ivan Prokipčák

Vedúci práce: Ing. Pavel Náplava

8. mája 2017

Poďakovanie

V prvom rade by som chcel poďakovať môjmu vedúcemu práce Ing. Pavlovi Náplavovi, za jeho odborné vedenie a cenné rady v oblasti koncepcie, Bc. Tomášovi Malinkovičovi za konzultácie a mentoring po technickej stránke a Mgr. Miroslavovi Dzurenkovi za praktické informácie v oblasti BPM. Veľká vďaka patrí aj mojim rodičom a najbližším, ktorí pri mne stáli a podporovali ma počas celej doby štúdia.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 8. mája 2017 .....

Odkaz na túto prácu

Prokipčák, Ivan. *Kvalitativní parametry webových aplikací na platformě IBM BPM*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Diplomová práca sa zaoberá optimalizáciou webových aplikácií na platforme IBM Business Process Manager, kde je na základe hypotézy formulovaná metodika, ktorej správnosť a univerzálna využiteľnosť je prakticky overená na zvolenom systéme.

Klíčová slova Business Process Manager, BPMN, metodika, najlepšie postupy, optimalizácia, webová aplikácia

Abstract

The aim of this diploma thesis is optimization of web applications on the IBM Business Process Manager platform, where the hypothesis formulates a methodology, whose correctness and universal usability is practically verified on the selected system.

Keywords Business Process Manager, BPMN, methodology, best practices, optimization, web application

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Po absolvovaní stáže v spoločnosti IBM Česká Republika, spol s r.o. ako BPM Consultant, v súčasnosti pracujúci na pozícii BPM Developer, som sa pri vývoji stretol s častým nepochopením či nedodržaním patternov a princípov, ktoré vedú k dobre udržiavateľnému a jednoducho rozšíriteľnému kódu pri vývoji procesných aplikácií.

Hlavným nedostatkom je chýbajúca dokumentácia a informácie zodpovedajúce aktuálnemu stavu BPM technológie. Každý nový vývojár znova naráža na problémy, ktoré už boli riešené, prípadne neprodukuje dostatočne kvalitný kód zodpovedajúci určitým konvenciám. Tento fakt spôsobuje že výsledné projekty sú po implementačnej stránke ťažko udržateľné a každé rozšírenie je čoraz komplikovanejšie. Vplýva to i na celkové fungovanie dodanej aplikácie a práca s takýmto systémom vedie k častej frustrácii zo strany užívateľa.

Na podnety skúsenejších vývojárov a neustále zvyšujúce sa množstvo problémov s rozšíriteľnosťou či udržateľnosťou väčších projektov som sa rozhodol v rámci diplomovej práce analyzovať problém, na základe toho vytvoriť hypotézu, ktorú ďalej formulujem do podoby metodiky a následne prakticky overujem v laboratórnych podmienkach.

Cieľom je, aby vzniknutá metodika a best practices boli uchopiteľné a univerzálne aplikovateľné naprieč celým spektrom projektov postavených na technológií IBM BPM a výsledné produkty boli dobre fungujúce z hľadiska developmentu i využiteľnosti z pohľadu užívateľov.

Štruktúra práce

V prvej kapitole rozoberiem kvalitatívne parametre webových aplikácií. Ďalšia kapitola popisuje frameworky a vzory, ktoré môžu kvalite webových aplikácií výrazne pomôcť, no nesprávnym použitím i pohoršiť. V trejtej kapitole rozoberiem problémy prevádzky. Predstaveniu zvolenej platformy BPM, analýze, rešerši možných riešení a hypotéze sa venujem v kapitole 4. Následujúcou kapi-

tolou predstavím zvolený systém, zanalyzujem jeho súčasný stav a navrhnem riešenie. V kapitole 6 transformujem hypotézu do podoby metodiky a v rámci kapitoly zároveň aplikujem v praxi na zvolenom systéme. Posledná kapitola popisuje testovanie a zhodnocuje prínosy či univerzálnosť využitia metodiky.

Kapitola 1

Web, webové aplikácie a ich vývoj

Dnes sa už len zriedka stretávame s vyhľadávaním informácií mimo prostredia internetu. Doba pokročila a ľudia neustále vyhľadávajú či vyvíjajú nové formy ako a kde informácie publikovať či efektívne získavať. K tomuto účelu boli spočiatku vhodné knižné podoby no súčasnosť patrí webovým stránkam a formám s nimi spojenými.

1.1 Definícia pojmov

Pri vývoji aplikácií či nástrojov spojených s prostredím internetu je nevyhnutné dobre pochopiť a vnímať rozdiely webu a webových aplikácií.

1.1.1 Webová stránka

Článok [6] uvádza, že webová stránka odkazuje na audiovizuálny obsah umiestnený na internete, ktorý je relatívne adresovaný formou URL. Väčšinou je zameraná na konkrétny produkt či firmu, za účelom zvýšenia predaja alebo zlepšenia reklamy a teda cieliť na konkrétnu skupinu návštevníkov. Jej obsah je statický a od užívateľa nevyžaduje zložitejšiu interakciu.

1.1.2 Webová aplikácia

Podľa [7] webová aplikácia obsahuje všetky prvky webovej stránky s tým rozdielom, že navyše je zo strany úžívateľa vyžadovaná určitá súčinnosť ako je napríklad zadávanie údajov alebo vykonávanie akcií, čo umožní následné generovanie dodatočného obsahu. Webová aplikácia spravidla komunikuje s užívateľom formou požiadavka a odpoveď a dáta sú spravidla udržiavané v databáze či lokálnom úložisku prehliadača.

Množstvo ľudí tieto prvky označuje súhrnne webom, čo nie je chybou a je to subjektívnym názorom každého zvlášť, no v oblasti vývoja sú tieto rozdiely podstatné najmä z pohľadu kvalitatívnych parametrov.

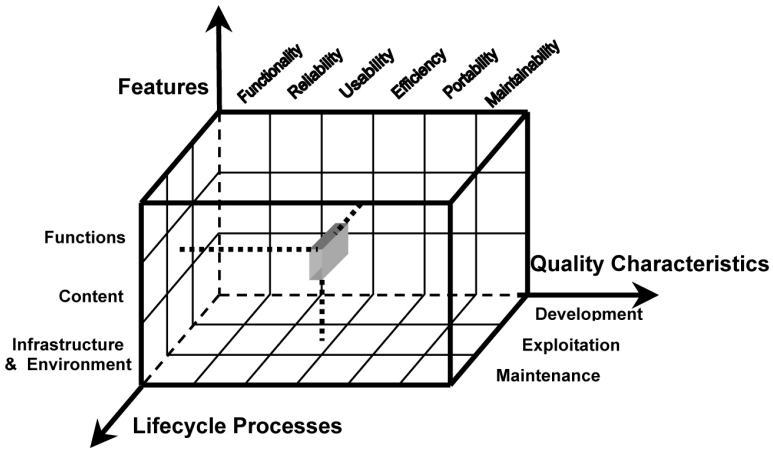
1.2 Kvalita webových aplikácií

V súčasnosti sme dospeli k štádiu, kedy samotná implementácia už nie je jediným problémom a predmetom analýzy. Rovnaké postavenie zaujal i účel, zmysel a kvalita vytvorenia produktu.

Teda podľa [8] už neriešime ako niečo vytvoriť, pretože technológie pokročili na úroveň, ktorá nám umožňuje prototypy produkovať v zlomku času potrebného na vývoj. Získavame tak hotové produkty, ktoré sú uchopiteľné a pre účely prezentácie postačujúce. Viac sa kladie dôraz na to či sa daný projekt vyplatí, čo prinesie a cieli sa viac na kvalitatívne parametre, keďže konkurencia a množstvo vzájomne si podobných projektov pribúda. Mnoho webov či webových aplikácii neplní účel alebo ich potenciál zostáva zbytočne nevyužitý.

Pre dosiahnutie optimálnych výsledkov aj z hľadiska kvality je dobré mať prehľad o dostupných nástrojoch či metodikách. Jedným z takýchto nástrojov je trojdimenzionálny model kvality, zachytený na obrázku 1.1.

Podľa článku [1] na ňom môžeme sledovať každý modelový rozmer: vlastnoti, kvalitatívne charakteristiky a procesy životného cyklu. Každý rozmer musí byť považovaný za hierarchickú štruktúru zloženú z ďalších základných prvkov.



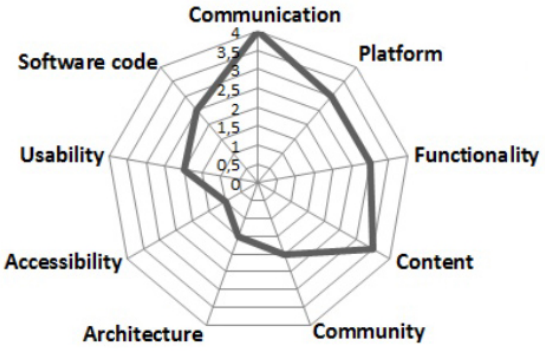
Obr. 1.1: Každá dimenzia vyjadruje inú skupinu kvalitatívnych parametrov[1]

Reprezentácia aplikácie v takejto forme nám dáva jasný prehľad o jej silných a slabých stránkach. Je už na prvý pohľad viditeľné, kde aplikácia nedo-

sahuje požadovanú kvalitu a vidíme, ktorú oblasť je potrebné zlepšiť. Každý atribut pozdĺž osy v sebe nesie splniteľnosť určitého kvalitatívneho parametra. Našim cieľom je, aby bola veľkosť objektu čo najväčšia, vyplnila obsah kocky a teda splňovala čo najviac požadovaných parametrov.

Z modelu možno vyčítať, že načrtnutý objekt dominuje po stránke vývoja, využiteľnosti a údržby. V porovnaní so zvolenou aplikáciou to bolo presne naopak, pretože tá má silné stránky hlavne z hľadiska funkcionality a obsahu, no vo výsledku sú pre nás dôležité všetky kritéria takže vzniká snaha akéhosi roztiahnutia objektu po celej 3D sieti.

Pre lepšiu predstavu je na obrázku 1.2 zobrazený 2D model, ktorý presne vypovedá o aktuálnom stave ZP, kde použiteľnosť (usability), architektúra (architecture) a softvérový kód (software code) sú aspekty, v ktorých je zlepšenie žiadúce.



Obr. 1.2: 2D model kvalitatívnych parametrov [2]

Po optimalizácií v niektorom z daných kritérií je potrebné diagram prekresliť, hodnoty merať opätovne pretože sa môže stať, že zlepšením jednej vlastnosti naopak zhoršíme inú. Tento nástroj nám ale umožní udržať si dostatočnú mieru nadhľadu.

1.3 Vývoj webových aplikácií

Tak ako som poznamenal, vývoj ostáva aj naďalej veľmi dôležitou súčasťou, i keď nie tak ako to bolo pred niekoľkými rokmi. Aktuálne technológie nám síce dávaju obrovskú moc a voľnosť, no aplikáciu zatiaľ samé nevyvinú, aj keď doba a nástroje čoraz viac smerujú k tomu, že hlboká znalosť niektorého z programovacích jazykov a programovania tak ako ho poznáme, už nebude potrebná. Viac sa dostáva do popredia analytické myslenie, návrhové vzory, prepojenie

či integrácia systémov a akýsi abstraktný pohľad na vývoj, kde dôležitú úlohu zohrávajú frameworky, ktorým je venovaná následujúca kapitola.

Kapitola 2

Frameworky

Na základe [9] v kontexte softvérového vývoja rozumieme framework ako súbor predpripraveného kódu či knižníc, ktoré poskytujú funkčnosť spoločnú pre celú triedu aplikácií.

Framework si môžeme predstaviť ako kostru, pevný základ, na ktorom môžeme budovať našu aplikáciu. Oproti knižniciam majú frameworky väčšie rozpätie funkcií, kdežto knižnice sa spravidla zemeriavajú na riešenie konkrétneho problému. Účelom framerworku je urýchliť vývoj tým, že nie je potrebné opätovne prepisovať funkcionalitu či štruktúru, ktorá sa v aplikácii bežne používa a teda vynachádzať koleso znova a znova. Oproti tomu dáva viac priestoru sústrediť sa na samotný vývoj a zamerať sa na podstatu riešeného problému.

Frameworky so sebou prinášajú aj určité obmedzenia a problémy. Medzi najzásadnejšie považujem skutočnosť, že komponenty frameworku sa neustále vyvíjajú, takže je potrebné vynakladať dodatočnú aktivitu pri udržiavaní funkčného stavu. Ďalším obmedzením sú licencie, ktoré komplikujú použiteľnosť a rozšíriteľnosť v prípade potreby. Napokon sú to i režijné náklady, ktoré sa môžu výrazne zvýšiť v závislosti na rozsahu využitia frameworku. Je teda vhodné analyzovať čo by mal framework riešiť a či sú jeho možnosti postačujúce, pretože každý má svoje špecifiká. Je potrebné si uvedomiť, že každý nástroj umožňuje v rukách experta dosahovať požadované výsledky veľmi efektívne, no je potrebné určiť si mantinely, aby doba vývoja zbytočne netrpela na nešťastne zvolenom frameworku. V následujúcich sekciách preto všeobecne popisujem situácie, pri ktorých riešení frameworky zbytočne obmedzujú, no aj tie, kde sú jednoznačným prínosom a zlepšením pri vývoji aplikácií.

2.1 Nevýhody

Schopnosť efektívnej práce s frameworkom je často získaná len dlhodobým vzdelávaním a intenzívnej práce v danej oblasti s konkrétnym nástrojom, inými slovami dlhou „learning curve“. Na druhú stranu, čím viac času človek venoval učeniu a pochopenie princípov, je osvojenie si znalostí práce s iným frame-

workom jednoduchšie a čas potrebný na dosiahnutie expertnej úrovne o to kratší.

V prípade, že sa vyskytne chyba, či už bezpečnostná alebo v rámci nestability systému, je to problém všetkých oblastí a projektov, kde bol framework použitý. Tento stav dáva útočníkom veľkú výhodu v prípade informácie o tom, kde všade bol framework použitý, získať prístup k citlivým dátam naprieč celým spektrom aplikácií.

Jasne definovaná architektonická štruktúra síce pomáha vývojárom udržovať si určité pravidlá a členenie funkcionalít, no naopak im berie slobodu a flexibilitu pri vyvíjaní aplikácie čo môže často znamenať nárast času potrebného k implementácii v princípe jednoduchej funkcionality do prostredia frameworku.

2.2 Výhody

Znovupoužiteľnosť kódu ktorý už bol raz napísaný, otestovaný a použitý inými vývojármi zvyšuje spoľahlivosť a znižuje čas celkového vývoja. Čas je jedným z hlavných ukazateľov a preto ak sme schopní redukovať množstvo času na vývoj, sme schopní znížiť cenu výsledného projektu, prípadne investovať ušetrené zdroje k dodatočnému rozvoju či optimalizácii aplikácie.

V prípade, že nevieme ako ďalej postupovať, je tu vždy možnosť informovať sa u skúsenejších. Je vysoká pravdepodobnosť, že rovnaký problém už niekto riešil, pretože frameworky sú väčšinou preferované širokou komunitou vývojárov a teda na to nikdy nie ste sami.

Správnym použitím frameworku sa výrazne zlepšuje produkt v rámci kvalitatívnych parametrov, ako sú použiteľnosť a efektívnosť. Zlepšuje sa i využite, údržba či vývoj v rámci dimenzie procesu životného cyklu. Zmeny je možno pozorovať i po stránke infraštruktúry či prostredia, čo v svojej podstate zabezpečuje už len samotné využitie frameworku, kde ide o vopred pripravenú a preddefinovanú štruktúru. Táto štruktúra je členená do funkčných celkov, ktoré nazývame vrstvy. Z toho plynie pojem viacvrstvá architekúra, ktorú popisujem v následujúcich častiach.

2.3 Viacvrstvá architektúra

Ako som poznamenal pri benefitoch frameworkov, ich štuktúra je založená na preddefinovaných architektonických vzoroch, ktoré dávajú kódu aplikácie určitú logiku a nezávislosť jednotlivých častí.

Voľba správnej architektúry systému je vždy vážnym problémom v procese vývoja softvéru. Veľké webové aplikácie nie sú výnimkou. Táto voľba má veľký vplyv na údržbu, rýchlosť či škálovateľnosť aplikácie.

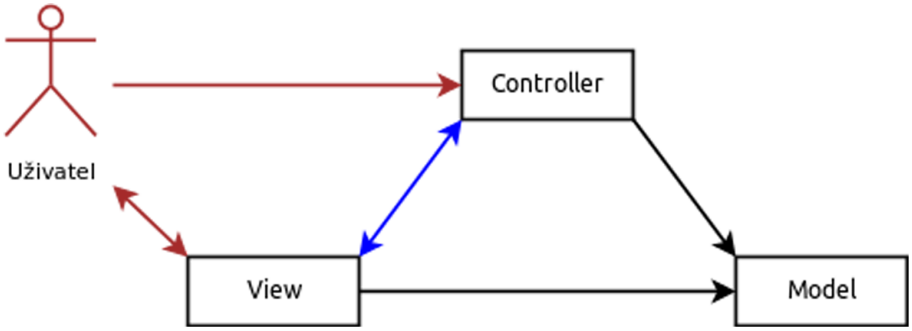
Medzi najčastejšiu štruktúru rozmiestnenia funkčných celkov do vrstiev patrí rozloženie Model-View-Controller (MVC).

2.3.1 Definícia MVC

V článku podľa [3] ide o základný návrhový vzor, ktorý umožňuje stavbu dobre štruktúrovanej aplikácie a vo výsledku minimalizuje závislosti medzi systémovými modulmi.

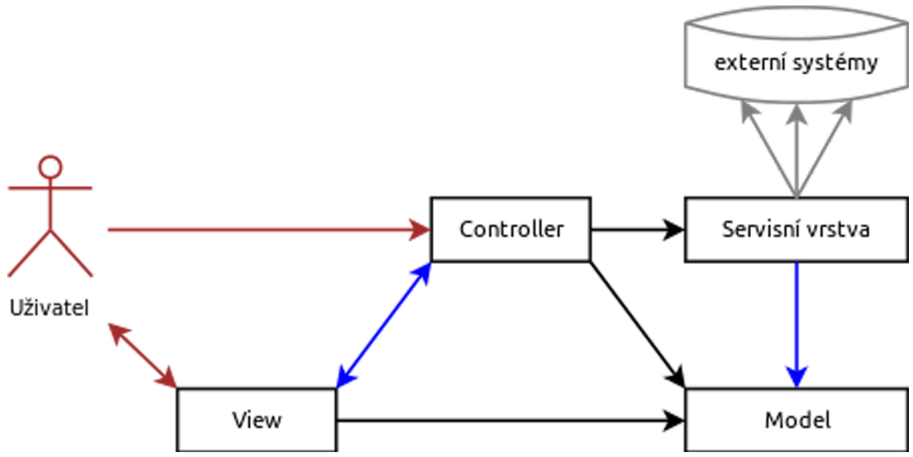
Model pozostáva zo statických a dynamických častí aplikačnej domény. Najdôležitejšou časťou modelu je aplikačná logika, ktorá obsahuje biznis logiku. Inými slovami, model špecifikuje dáta a správanie aplikácie, s konkrétnou prezentáciou nemá nič spoločné. View obsahuje priamy odkaz na model a je teda zodpovedný za grafickú alebo textovú reprezentáciu jeho dát. Je to serverový kód, ktorý generuje informáciu následne zobrazovanú konečným užívateľom napr. formou HTML kódu.

Controller plní úlohu akéhosi radiča, ktorý riadi tok udalostí v programe – aplikačnú logiku. Taktiež je priamo spojený s modelom, kde pomocou implementovaných funkcií modifikuje jeho dáta, ale aj s view, kde na základe nejakej akcie od užívateľa v ňom môže priamo zmeny zobraziť.



Obr. 2.1: Grafická podoba vzoru MVC [3]

Ak je ale potrebné, aby program komunikoval s vonkajším svetom, napr.: s databázou, je vhodné architektúru rozšíriť o modul, ktorý zabezpečí a poskytne danú súčinnosť i ostatným častiam systému. Z návrhu architektúry vyplýva, že modul nesmie byt súčasťou modelu ani ktorejkoľvek inej vrstvy, a preto sa do vzoru pridáva ako vrstva samostatná. Takúto vrstvu nazývame servisná.



Obr. 2.2: Grafická podoba vzoru MVCS [3]

Podľa článku [10] je pridaná hodnota použitia MVC založená na dvoch základných pravidlách a to oddelenia modelu – štruktúry, a časti prezentačnej, čiže presenteru. S týmto rozdelením je napríklad možné kedykoľvek zmeniť užívateľské rozhranie bez potreby väčších zásahov do štruktúry a kódu ostatných vrstiev aplikácie. Druhým pravidlom je oddelenie vrstvy view (V) a controlleru (C). Typickým príkladom je práca s dvomi kontrolermi (editovateľný a needitovateľný) pre jeden view.

So spomínanými i mnohými ďalšími návrhovými vzormi pracujú všetky pokročilé frameworky obzvlášť IBM BPM (IBPM), ktorému sa budem v tejto práci venovať podrobne a rozoberiem jeho možnosti i postupy vývoja.

2.3.2 Design patterny

V súvislosti s návrhom existujú mnohé postupy, návrhové vzory, ktoré nám pri zavedení poskytujú spomínanú mieru abstrakcie. Podľa [11] predstavujú osvedčené postupy a najlepšie riešenia, ktoré by mali byť použité pri vývoji. Sú riešením všeobecných problémov, na ktoré v priebehu vývoja počas dlhého obdobia narazili mnohí.

Návrhové vzory poskytujú štandardnú terminológiu a sú špecifické pre konkrétny scenár. Jednoduchý dizajnový vzor znamená napríklad použitie jedného objektu tak, že všetci vývojári sú oboznámení s jeho designom a budú ho využívať v danej súvislosti na riešenie konkrétneho prípadu.

Vo všeobecnosti je známych celkom 23 návrhových vzorov, ktoré sú členené do podskupín na Creational Patterns (vytváracie), Structural Patterns (štrukturálne), Behavioral Patterns (správanie). Každý je jedinečný a rieši určitú problematiku.

Medzi najznámejšie patrí napríklad Dependency injection (ďalej DI), ktorý rieši predávanie závislosti medzi jednotlivými komponentami aplikácie tak, aby sa komponenty mohli vzájomne používať. Môžeme chápať formou akéhosi parametru, na ktorom je beh triedy či objektu závislý. Ten kto chce našu triedu použiť, musí pre korektné správanie hodnotu dodať. To ako ju získa neriešime my ako autor, ale jej používateľ – vývojár. Na tomto princípe šú postavené všetky komponenty v IBM BPM, čo demonštrujem pri formulovaní metodiky.

Ďalším príkladom je vzor Adaptér. V [12] sa píše, že jeho využitie oceníme pri práci s komponentami, ktoré nám poskytujú nekompatibilné rozhranie s našou aplikáciou.

Ako praktický príklad slúži vytvorenie služby, ktorá pracuje s externým rohraním KOSu, ktoré ale vracia dáta v nekompatibilnom formáte. Preto vzniknutá služba zabezpečuje, že všetko čo rozhranie KOSu vráti, spracuje na dátové objekty, s ktorými je možné v BPM pracovať, prípadne dáta z BPM transformuje do formátu zrozumiteľnom pre KOS. V takomto prípade sme vo všeobecnosti nútení vytvoriť triedu, resp. zaobaliť stávajúcu naším rozhraním, ktorá nám túto funkcionalitu zabezpečí. V praxi sa často stretávame s komponentami tretích strán a teda využívaním už hotových riešení. Predstavme si ale, že využitie danej časti je pre nás kľúčové, no vývoj komponenty, s ktorou pracujeme ešte nie je ukončený. Problém nastáva, keď je komponenta použítá vo viacerých častiach systému, niečo sa zmení, systém s veľkou pravdepodobnosťou prestane fungovať a fixácia problému zaberie značný čas. Vzor adaptér komponentu obalí naším rozhraním a teda aplikácia je odstienená od pôvodného rozhrania komponenty. V prípade potreby či aktualizácie pracujeme už len s adaptérom. Vďaka tomuto vzoru môžme bezproblémovo riešiť i situáciu, kedy komponentu úplne zmeníme. Vtedy znova jednoducho upravíme adaptér a chod aplikácie to neovplyvní.

Zavedenie a hlavne udržiavanie vzorov nie je jednoduché no pre dobrú udržateľnosť a prehľad nevyhnuté. V závere je ale vývojár odmenený dobre udržiavateľnou štruktúrou, kde nedochádza k použitiu nadbytočného či opakovaného kódu a plní svoj účel vzhľadom k zapojeniu modulov vyvíjaných paralelne iným týmom ľudí nezávisle od jadra systému.

Kapitola 3

Problémy prevádzky

Náklady na hardvérové a softvérové vybavenie sú priamo úmerné objemu návštevníkov daného servera. Je nutné si uvedomiť, že bez dostatočnej prípravy na rastúcu popularitu webovej aplikácie, začne aplikácia a jej návštevnosť pomaly klesať. [13] Pri zvolenom systéme sa o návštevnosť nemusíme báť, keďže ide o nástroj nevyhnutný k ukončeniu či postúpeniu v rámci štúdia. Poslaním je ale čo najlepšie pripraviť aplikáciu pre väčšie množstvo užívateľov a zabezpečiť ich spokojnosť pri práci so systémom. Na kvantifikáciu používateľskej spokojnosti slúži koncept „kvalita služby“ (Quality of Service, QoS)[14]. V tejto práci sa pokúsim rozobrať a zamerať sa na čas odozvy (response time). Spravidla sa tento parameter medzi užívateľmi líši. V rámci optimalizácií aplikácie sa pokúsim tento stav maximálne vylepšiť a pri testovaní objektívne zhodnotiť.

3.1 Testovacie metriky

Na popis používania webovej aplikácie existuje mnoho metrík. Tieto metriky je nutné brať do úvahy počas celého priebehu testovania. Zmyslom toho všetkého je realisticky vyhodnotiť, aké parametre bude mať bežná záťaž testovanej aplikácie, a podobnú záťaž umelo generovať.[13] V rámci testovacej časti sa zameriam na odozvu a rýchlosť načítania stránok.

Kapitola 4

Vývoj v IBM BPM

Účelom tejto kapitoly je definovať technológiu IBM BPM a podrobne sa zaoberať jej možnosťou využitia pri vývoji webových aplikácií. Popisovaná technológia je používaná i v rámci praktickej časti, na ktorej je zvolený systém postavený.

4.1 Pojmy

Pred analýzou je nutné oboznámiť sa s týmito pojmami.

4.1.1 IBM WebSphere

IBM [15] definuje IBM WebSphere ako softvérovú platformu firmy IBM založenú na architektúre SOA poskytujúcu prostredie pre integráciu systémov, aplikácií a služieb.

4.1.2 BPMN

Je podľa [16] definovaná ako sada grafických prvkov, ktorá prepája dizajn obchodných procesov s jej implementáciou. V kontexte s IBM BPM ide o súbor objektov, kde každý v sebe nesie určitú funkcionalitu a s ich pomocou je možné v BPM frameworku namodelovať proces na najvyššej úrovni.

4.1.3 Toolkit

Je podľa Kolbana [17] podobný procesnej aplikácii. Toolkit môže byť tiež považovaný za kontajner pre komponenty používané v riešeniach. Na rozdiel od procesnej aplikácie Toolkit neobsahuje nasaditeľnú aplikáciu. Namiesto toho obsah Toolkitu môže byť "zahrnutý" alebo "použitý" jednou alebo viacerými procesnými aplikáciami

4.1.4 Coach View

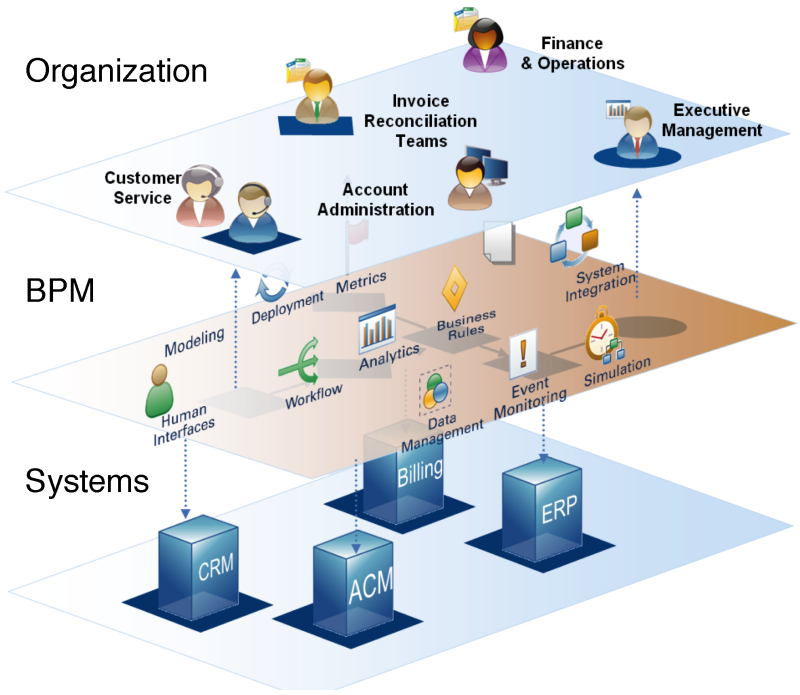
Kolban [17] píše, že všetky stránky v IBPM obsahujú vizuálne stavebné bloky, ktoré spolu tvoria jej obsah. V IBPM sa každý z týchto stavebných blokov nazýva "Coach View"

4.1.5 Human Service

Podľa Kolbana [17] je najdôležitejším typom služby v IBPM. Je to komponenta, ktorá ak je spustená, vytvára pre človeka úlohu, na ktorej sa dá pracovať. Proces je v tomto prípade pozastavený až do ukončenia tejto úlohy. HS v sebe zahrňuje CV komponenty, slúžiace k popisu dát na obrazovke a sú užívateľom prezentované, ako i dáta, ktoré majú byť od užívateľov požadované.

4.2 Predstavenie IBM BPM

IBM [4] definuje IBM Business Process Manager (IBPM) ako sadu nástrojov, viď obrázok 4.1, pre správu obchodných procesov. IBPM poskytuje platformu na ktorej môže byť business proces definovaný, implementovaný, spustený a monitorovaný.



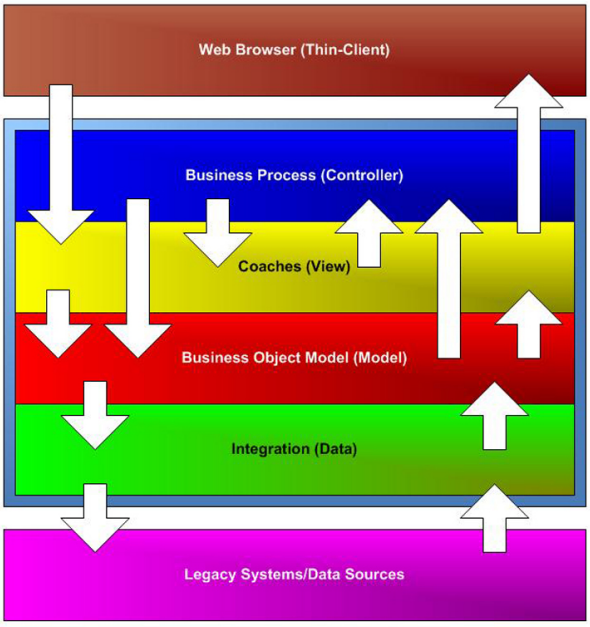
Obr. 4.1: Sada nástrojov IBPM[4]

4.2.1 Benefity IBM BPM

Zdroj [17] definuje, že obchodné procesy obvykle existujú v mysliach a praxi zamestnancov, no je veľmi náročné ich premietnuť a zabezpečiť, aby boli pochopené i zo strany vývojárov. Implementácia, ktorá je jednou z hlavných výhod IBPM, umožňuje výrazne skrátiť trvanie porozumenia prostredníctvom interaktívnych zasadnutí, kde je proces priamo zachytený v IBPM ako intuitivný diagram. Po "náčrte" procesu je možné proces spustiť a umožniť tak stranám, ktoré rozumejú jeho povahe, aby potvrdili, že to, čo bolo implementované je správne. Pokiaľ sú nájdené chyby, je možné proces jednoducho opraviť až pokým nebude správne zachytený a pochopený i zo strany vývojárov. Tento interaktívny a realtime cyklus je vďaka IBPM veľmi flexibilný čo je hlavný rozdiel a benefit pri zrovnávaní s podobnými produktami na trhu.

Ďalšia, nemenej dôležitá vlastnosť je integrácia. Množstvo obchodných procesov v sebe zahrňuje iba prenášanie úloh z jedného užívateľa na druhého, no majorita systémov vyžaduje integráciu či vzájomnú komunikáciu. IBPM poskytuje schopnosť vnímať externé systémy ako časti procesov. Samotná integrácia je potom realizovaná formou webových služieb alebo iných komunikačných volaní a to všetko v rámci jednej platformy, prípadne procesu.

Pre lepšiu predstavu rozloženia vrstiev prikladám následujúci diagram 4.2



Obr. 4.2: Grafická podoba vrstiev v IBM BPM[5]

4.3 IBM BPM v praxi

V súčasnosti je technológia zabehnutá a nasadená na viacerých projektoch v bankovom, poisťovníckom či školskom sektore. Prípadné detaily nie sú určené verejnosti a preto sa v práci zamerám len na využitie vo zvolenej aplikácii, ktorú predstavím v následujúcej kapitole.

4.3.1 Problémy pri vývoji v IBM BPM

IBPM technológia je veľmi robusným nástrojom čo so sebou nesie množstvo problémov. Jedným z nich je ten, že v komunite vývojárov sa nachádza len zlomok tých, ktorí ju vedia využívať so všetkým čo ponúka. Ostatní a začínajúci vývojári sa musia znovu trápiť pri riešení problémov, ktoré už boli, spravidla i lepšie, vyriešené. IBPM technológia je pomerne mladá, no pomaly nastáva čas kedy je potrebné informácie centralizovať a umožniť tak jednoduchšie predávanie informácií.

I keď samotný IBPM núti od samého začiatku členiť implementáciu do logických vrstiev, ako je napríklad procesná, servisná či frontendová, vývojár je vedený k produkovaniu dobre udržiavateľného kódu. Je ale nutné poznamenať, že v praxi sa i napriek tomuto stáva, že výsledná implementácia vôbec nezodpovedá dobrým mravom a konvenciám, ktoré by mali byť hlavnou zásadou pri vývoji akejkoľvek aplikácie.

Je to spôsobené najmä tým, že nástroj umožňuje i napriek zdanlivo jednoznačným nástrojom, nastaveniam či parametrom množstvo ciest ako sa dostať k požadovanému výsledku. Je preto dôležité stanoviť si kritéria, či vzory, ktorými sa bude tým vývojárov riadiť a teda údržba či budúci rozvoj systému nebude výzvou.

Tento problém som sa pokúšal vyriešiť v rámci rešerše, nájdením dokumentácií či obdobných súborov najlepších praktík v rámci podobných technológií.

Vo výsledku sa mi nepodarilo nájsť žiadne všeobecné materiály, ktoré by mohli byť vnímané ako hotová metodika, náčrt či inšpirácia riešení určitých situácií, no čiastočné informácie obsahujú publikácie spoločnosti IBM, ktorým sa venujem v rámci rešerše v následujúcej sekcii.

4.4 Rešerš, best practices

Keďže mám možnosť, ako jeden z vývojárov, pôsobiť v komunite BPM, naskytla sa mi jedinečná možnosť informovať sa priamo od ľudí, ktorí túto technológiu posúvajú ďalej. Všetky odpovede boli jednoznačné a viedli k záveru, že podobný výukový materiál neexistuje. Boli mi ale odporúčané materiály, ktoré môžu poslúžiť ako inšpirácia.

4.4.1 IBM Redbook

Tento materiál by v sebe mal, podľa názvu [18] "Performance Tuning and Best Practices", zahrňovať všetko, čo potrebujeme. V tom prípade by nebolo potrebné, aby vznikal podobý materiál, keďže kniha je tvorená samotnou IBM a ľuďmi, ktorí sa na vývoji platformy BPM priamo podieľali.

Po dôkladnom prejdení sa obsah kapitoly 3 podobá konceptu, ktorý sa formuluje v rámci tejto práce. Nedostatkom je, že tam aj tento zámer končí a ďalej sa optimalizáciou a riešením chýb zaoberá len textovo a teoreticky bez praktických ukážok, čo je podľa mňa veľkým nedostatkom. Nachádza sa tam síce množstvo užitočných rád a konfigurácií, no menej znalý vývojár si z toho prakticky nič neodnesie. Je ale určite vhodné knihu používať ako doplňujúci materiál.

4.4.2 Kolban's Book on IBM BPM

V súčasnosti je [17] "Kolban's Book" jediný zdroj, ktorý v sebe centralizuje všetky potrebné informácie k získaniu základných znalostí potrebných pre prácu s IBPM. Optimalizáciu a najlepšie praktiky síce nerieši, a keď tak okrajovo, dáva ale všeobecný rozhľad čo technológia IBPM umožňuje a rady ako s ňou pracovať. Po osvojení si informácií obsiahnutých v tejto knihe, je vývojár prakticky pripravený s možňosťou využitia na reálnom projekte.

Zdroj v sebe obsahuje množstvo obrazového materiálu, čo len umocňuje rýchlosť pochopenia a použitia znalostí. Knihu považujem ako inšpiráciu po grafickej stránke zobrazenia obsahu, ktorej by som sa chcel držať v kapitole pri hľadaní riešenia.

4.4.3 Vyhodnotenie hľadaní

Výsledky hľadaní teda jednoznačne potvrdzujú, že materiály s podobným obsahom, sa v podobe, ktorá by združovala informácie a jednoznačne formulovala rady ohľadom optimalizácie, zatiaľ neboli vytvorené. Na tento záver naviažem v nasledujúcej kapitole zhromaždením pravidiel a praktických rád, ktorými by sa mal vývojár pri produkovaní dobre udržateľného kódu držať, prípadne vzorových situácií, ktorým by sa mal naopak vyhýbať. Tieto poznatky sú získané od ľudí, ktorí ich získali dlhoročnou praxou v oblasti vývoja procesných aplikácií.

4.5 Hypotéza

Na základe poznatkov zozbieraných v predchádzajúcich kapitolách sa domnievam, že vytvorenie vhodného súboru pravidiel by mohlo slúžiť ako užitočný nástroj k zlepšeniu kvality webových aplikácií postavených na platforme BPM.

Preto som pre praktickú časť stanovil hypotézu, že fomuláciou pravidiel do podoby metodiky a následne aplkáciou dobrého metodického konceptu na zvolený systém je možné, dosiahnuť zlepšenie kvalitatívnych parametrov vo všetkých dimenziách.

Kapitola 5

Predstavenie systému a stav

Praktická časť a overenie hypotézy je vykonané na aplikácií Záverečné práce (ZP). Systém ZP je webová aplikácia vyvinutá skupinou CZM, pôsobiacou pod FEL ČVUT. Je postavená na platforme IBM BPM a bola vytvorená za účelom digitalizácie procesu záverečných prác pre FIT ČVUT. Aplikácia v sebe zahrňuje celý proces od vytvorenia témy ZP vedúcim, rezervácie študentom cez schvaľovací proces príslušníkmi vedenia fakulty, odovzdávania, až po oponentúru a zadanie výsledného hodnotenia. Na pozadí beží webový portál, ktorý jednotlivé úkony a posuny v procese delí a priradzuje na základe rolí čím zabezpečuje, že každý pracuje len s úlohami, ktoré sú mu určené. Aplikácia sa v súčasnosti nachádza v produkcii a je v stabilnom a plne funkčnom stave, no spôsob akým je implementovaná je z pohľadu výkonu, rozšíriteľnosti či udržateľnosti kritický.

5.1 Súčasné problémy

Hlavným nedostatkom je rýchlosť pri práci so systémom z pohľadu užívateľa, kde je doba odozvy neznesiteľne vysoká. Je to spôsobené zlou optimalizáciou či množstvom nadbytočných požiadavkov oproti databáze. Tieto chyby neboli spôsobené zlým návrhom, ale častými úpravami a zmenami samotného jadra aplikácie. Prispelo k tomu i veľké množstvo dodatočných požiadavkov, pri ktorých sa kládol dôraz primárne na rýchlosť implementácie namiesto kvality kódu a optimalizácie, teda korektného vnesenia novej funkcionality i z pohľadu architektonického. S týmto problémom súvisi i rozšíriteľnosť, ktorá je v súčasnom stave aplikácie síce možná, no veľmi časovo náročná. Mnoho komponent sa zbytočne znova definuje, nie sú dobre štrukturované, ťažko sa vyhľadávajú takže každá neuvážená zmena len komplikuje aktuálnu situáciu.

5.2 Riešenie problému

Spomínané problémy sa za behu riešia veľmi ťažko. Mnohokrát je lepšie znovu analyzovať a od začiatku implementovať. Ja sa ale pokúsim o zmeny, ktoré by tento stav vylepšili a v lepšom prípade i napravili. Pôjde o riešenia problémov združených do súhrnných celkov, kde každý z nich bude vypovedať o určitom type optimalizácie. Každý celok bude popisovať najčastejšie problémy, ktoré sa vyskytli v zvolenom systéme, analyzovať riešenie, ktoré sa následne pokúsim aplikovať na zvolený systém. Riešenie následne zovšeobecním, aby bolo použiteľné i v rámci iných projektov.

Vo výsledku pôjde o formuláciu a následnú aplikáciu súboru pravidiel, najlepších praktík a vzorov, ktoré by mali byť dodržiavané pri vývoji akejkoľvek aplikácie na platforme IBPM.

Kapitola 6

Optimalizácia a formulácia metodiky

Samotný pojem optimalizácia v sebe zahŕňa široké spektrum rôznych kritérií. Pri zmenách v rámci mojej časti implementácie som sa zameral na optimalizáciu procesného flow aplikácie ZP a teda sprehľadnenia a zjednodušenia možnosti dodatočného rozvoja aplikácie. Upravoval som procesy na úroveň jednoduchého pochopenia a zjednodušoval zbytočne zložité časti. Ďalej som vykonával optimalizácie vzhľadom k logom a odchytávaniu chybových stavov. Napokon som sa zameral na frontendovú časť a to ako z hľadiska rozloženia prvkov (komponent) tak i nadbytočného volania AJAXových funkcií pri vykresľovaní jednotlivých častí. Pri zmenách štruktúry som na základe starých, vytváral nové lepšie komponenty, ktore som ďalej členil do Toolkitov.

Táto kapitola sa zaoberá súhrnom najlepších postupov, ktorých implementácia a dodržiadavnie zabezpečujú plynulý chod aplikácie či už z pohľadu užívateľov alebo jednoduchosti pre vývojárov v rámci rozšíriteľnosti. Postupy sú členené na základe vrstiev, ktoré v stebe zapúzdrujú rôzne pohľady a funkcionalitu a sú to tieto tri:

- Procesná vrstva - definuje logiku a proces, ktorým sa aplikácia riadi a na jeho základe postupuje. Prvky sú na ňom definované formou BPMN a umožňujú modelovanie požadovaného správania na najvyššej (procesnej) úrovni.

- Frontendová vrstva - definuje komponenty, ktoré su viditeľné užívateľmi aplikácie. Nachádzajú sa v nej Coach Views (CV), ktoré sú buď súčasťou ďalších, väčších CV alebo vystupujú samostatne.

- Servisná vrstva - nesie v sebe backendovú logiku a všetko s ňou spojené. Je modelovaná taktiež formou BPMN, no komponenty sú viac orientované na spracovávanie dát, komunikáciu s databázou či inými externými zdrojmi.

Medzi najčastejšie chyby, ktoré sa vyskytujú naprieč všetkými vrstvami patria následujúce. Pri popise problémov v aplikácii som narazil i na ďalšie, ktorým sú venované samostatné sekcie. Je potrebné si dávať pozor hlavne na tieto chyby:

- Redukovať množstvo komponent (krabičiek), ktoré v sebe nesú volania databáze. Niekoľko násobné volanie v rámci jednej systémovej služby je zbytočné a zhoršuje odozvu i rýchlosť systému. Je dôležité zvážiť, či pridanie ďalšej krabičky – volania má nejaký význam a nie je lepšie vykonať zmeny rovno v integračnej službe. Takto vytvorené služby je potrebné optimalizovať.

- Málo zdokumentovaný kód, či už ide o premenné, funkcie alebo rovno celé bloky. Budúci vývojár tak zbytočne stráca čas, ktorý potrebuje k pochopeniu kódu či zdĺhavému zisťovaniu účelu už implementovanej funkcionality.

- Pred tvorením novej časti, či už je to premenná, funkcia alebo rovno služba je potrebné najprv dôkladne analyzovať, aby nevznikali zbytočné duplicity, ktoré taktiež sťažujú orientáciu, prípadne optimalizáciu. Preto je potrebné zjednotiť pomenovania a poradie premenných.

- Naopak pri odstraňovaní existujúcich častí či už sú to premenné, funkcie až zložité časti kódu, platí dôležitosť analýzy dvojnásobne. Niekedy odstránenie navonok nepotrebnej premennej môže mať obrovský dopad na funkcionalitu aplikácie. Z praxe viem, že logické chyby sa hľadajú najhoršie, keďže k ich nájdeniu a odstráneniu je potrebné kódu a jeho štruktúre dobre rozumieť, čo je v prípade človeka, ktorému nie je daný kód vlastný, veľmi časovo náročný problém. Najlepšou radou je nepremenovávať ani nemazať nič pri čom si nie som istý účelom. Problém sa odporúča poznačiť a riešiť až v prípade väčších zmien v štruktúre, tzv. refaktoringu.

- Zhlukovanie kódu už bolo spomenuté v časti o duplicitách a opätovných volaniach rovnakých služieb s inými parametrami. V tomto pravidle dávam do pozornosti zjednotenie v rámci inicializácie, kde sa často stáva, že objekt či premenná sa inicializuje až v časti, kde sa používa - volá. Ak je ale premenná volaná niekde v kóde pred, nastáva problém, kde siahame na miesto, ktoré nie je definované, čo spôsobí chybu až pád aplikácie. Najlepším riešením je inicializácie centralizovať v rámci jedného inicializačného objektu, ktorý voláme vždy ako prvý, pred použitím akejkoľvek inej funkcionality.

6.1 Procesná vrstva

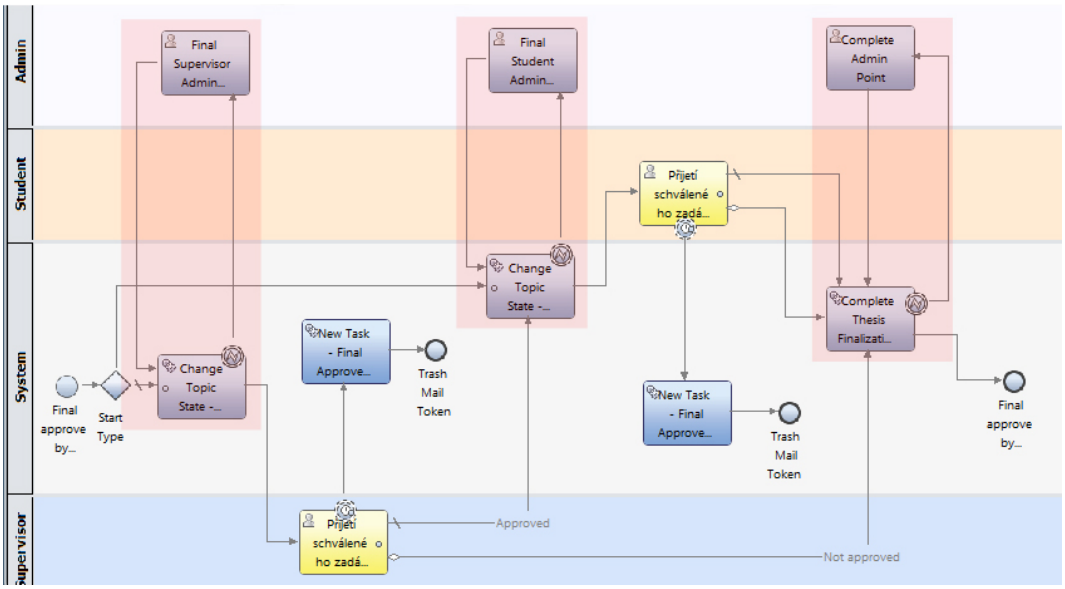
Optimalizácia v rámci procesnej vrstvy spočívala v úplnom pochopení fungovania celkovej aplikácie, jednotlivých stavov či vetvení, pretože každá, zdanlivo drobná, zmena na tejto úrovni má značný vplyv na funkcionalitu celej aplikácie. Užívateľ sa tak nečakane môže dostať do stavu, ktorý preňho nie je korektne definovaný (aplikácia napríklad očakáva dáta, ktoré daný objekt neobsahuje) a teda spôsobí nechcený pád, v lepšom prípade zlú reakciu.

6.1.1 Logovanie a odchytávanie chybových stavov

Logy sú jediným zdrojom informácií, ktorými vývojár získava prehľad o celkovej činnosti užívateľov v aplikácií. Je to zdroj, ktorý poskytuje informácie o udalosti vykonávanej pred pádom či neočakávaným výstupom. Preto by sa k tejto činnosti malo pristupovať zodpovedne a každý celok či komponentu, pri ktorej hrozí riziko pádu alebo dôkladne ošetriť a zmapovať zavolaním logu a pridaním odchytávania výnimky.

6.1.1.1 Problém

Na obrázku 6.1 na prvý pohľad vidíme neprahľadnú štruktúru a rozmiestnenie procesov. Značne k tom prispieva i "error a exception handling", ktorý je realizovaný odvedením procesu do časti, na obrázku 6.1 pomenovanej, "Admin Point".

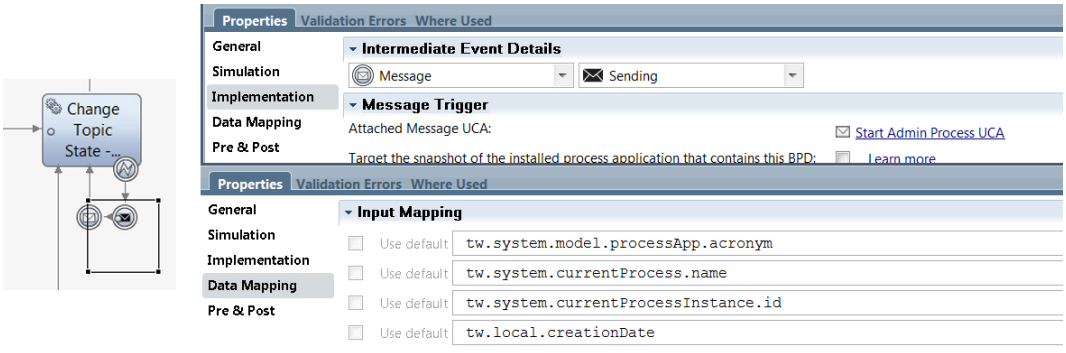


Obr. 6.1: Nadbytočné komponenty procesu s názvom "Admin Point"

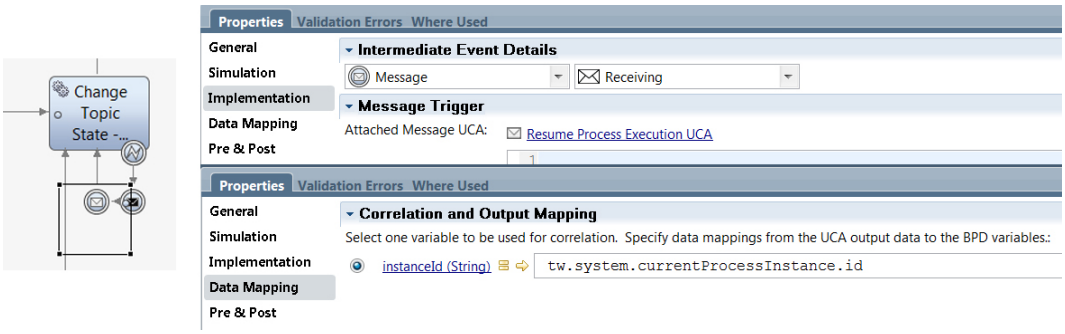
6.1.1.2 Riešenie

V prvom rade zadefinujem používaný pojem Undercover Agent (UCA), čo sú podľa Kolbana [17] komponenty v IBPM, ktoré naslúchajú na udalosti založené na plánovanom čase. Sú takisto zodpovedné za spúšťanie procesných inštancií, resp. reakciu na prichádzajúce požiadavky, prostredníctvom správ odosielaných inou časťou či procesom samotným. Spracovávanie chýb sa odporúča riešiť cez komponenty nazývané "Intermediate Eventy" zapojením tak ako vidíte na obrázku 6.2. Takéto zapojenie sa vo vývojárskom slangu nazýva "koloběžka". Ďalej sa odporúča vytvorenie samostatnej aplikácie s názvom napríklad "Admin Application", obsahujúca proces, ktorý sa daným chybovým eventom napojenom na komponente spúšťa. Teda UCA odoslaným aplikáciou, v ktorej chybový stav nastal. Štart procesu je vlastne reakcia na prichádzajúce správy typu chyba.

Použitie a konfigurácia vstupných premenných je zrejmá z následujúcich obrázkov 6.2 6.3.



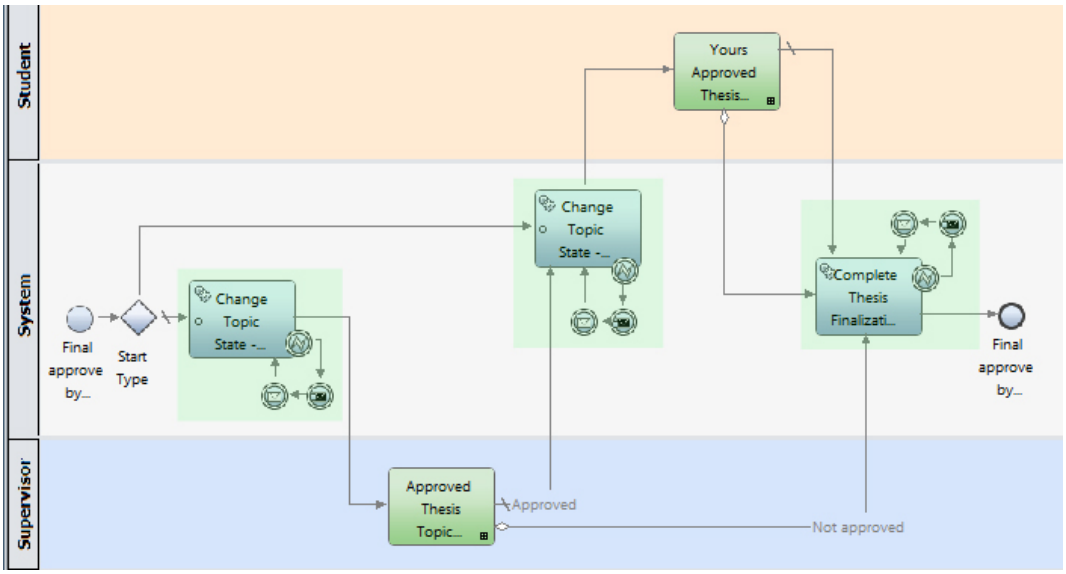
Obr. 6.2: Pri chybovom stave sa odošle správa (formou UCA) na administrátora, kde sa mu vytvorí v procesnom portále nová úloha



Obr. 6.3: Po spracovaní, náhľade chyby a stlačení tlačítka sa odošle správa, na ktorú reaguje príslušný event

V administrátorskej aplikácii je možné vytvoriť pre správu prakticky akúkoľvek užívateľsky prívetivú aplikáciu. Postačí ale jednoduchý Coach, ktorý zobrazuje informácie o chybe a nejaký element (tlačítko), ktorým užívateľ presunie proces zo stavu “pozastavený“ (nie je možné s ním pracovať) späť do stavu “aktívny“ (pôvodný stav, pred tým ako nastala chyba) v prípade, že je všetko v poriadku. Prístup k takémuto tasku má spravidla formou procesného portálu úžívateľ s aministrátorskými právami.

6.1.1.3 Prínos metodiky



Obr. 6.4: Výsledná komponenta po aplikácií systému odchytávanie výnimiek formou "koloběžky"

Ako z problému vyplýva, jednoznačným prínosom je zvýšenie prehľadnosti procesu, ako je možné vidieť na obrázku 6.4, ba dokonca odpadá i nutnosť administrátorskej swimline, ktorá je zabezpečená tým, že UCA sú odosielané na aplikáciu, ktorá v sebe danú rolu obsahuje a teda my sa o túto skutočnosť nemusíme starať.

6.1.1.4 Zhrnutie vo všeobecnosti

Takýto systém spracovania je vhodný a aplikovateľný na akúkoľvek aplikáciu. Stačí len vytvoriť všetky potrebné celky a naviazať na časť procesu, kde sa chyba môže vyskytovať. Takáto funkcionalita často nebýva súčasťou aplikácie ale je používaná formou toolkitu.

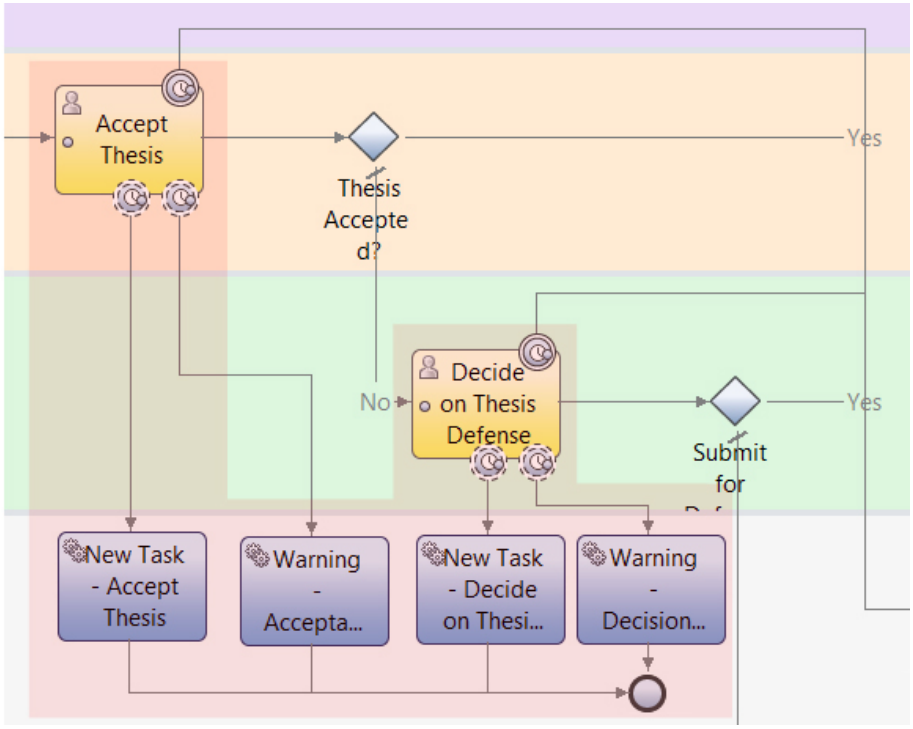
6.1.2 Členenie logických celkov do subprocesov

Jedným z dôležitých úkonov, ktoré prispievajú k lepšiemu prehľadu v procesoch je zoskupovanie súvisiacich častí do subprocesov. Pri takýchto úpravách procesu je dôležité si uvedomiť, ktoré časti sú si príbuzné a majú medzi sebou súvis.

Ak sa toto kritérium nedodrží, tak vývojár môže mylne a zdĺhavo, napokon neúspešne, hľadať konkrétnu časť či funkčný celok procesu, čo zbytočne navyšuje čas potrebný k rozvoju a teda tento problém uberá na účinku samotnej optimalizácie z pohľadu rozšíriteľnosti.

6.1.2.1 Problém

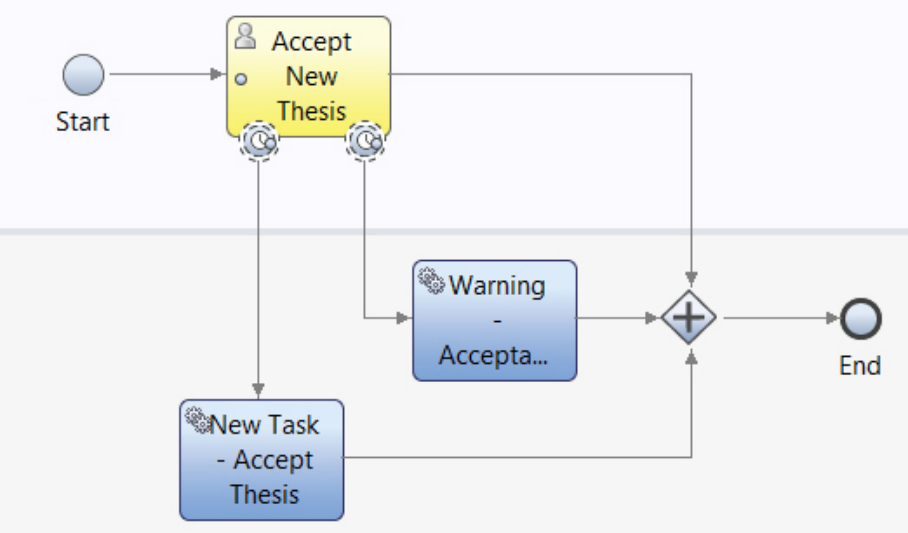
Problém vzniknutý v tejto časti je zrejmý z obrázka 6.5, kde vidíme zbytočnosť zobrazenia funkcionality a volaní služieb. Je to názorný príklad, ako by to nemalo vyzerať a na konci i vidieť ako výrazne sa zlepší prehľadnosť procesu aplikáciou pravidla členenia do subprocesov, viď obrázok 6.7.



Obr. 6.5: Pôvodný stav procesu a jeho zbytočne viditeľné časti

6.1.2.2 Riešenie

Vytvorením podprocesu, do ktorého danú funckionalitu 6.6 zapúzdrime a správame sa k nej ako k samostatnému "mini procesu", čo je vidieť na obrázku 6.7

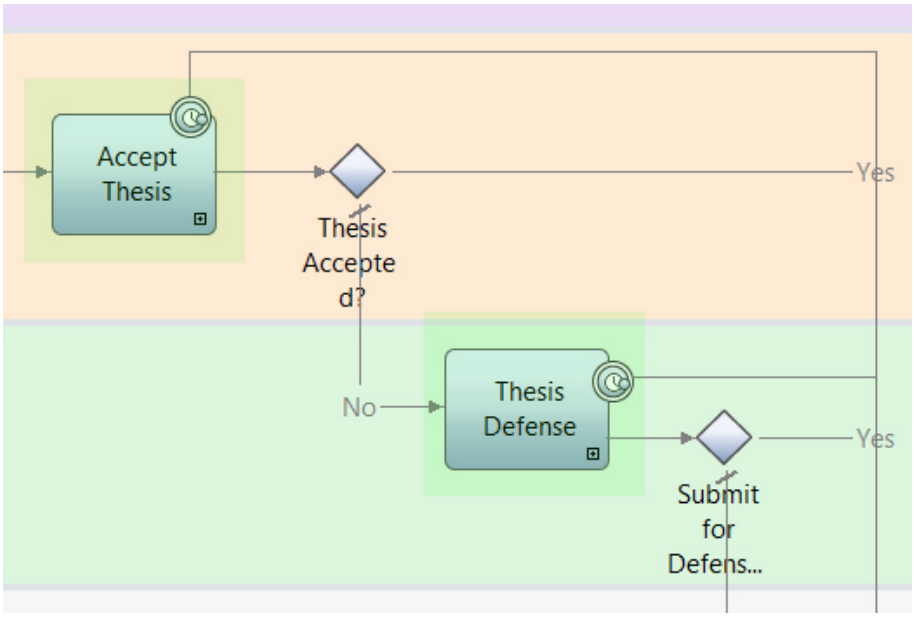


Obr. 6.6: Časť procesu presunutá pod samostatný subproces

6.1.2.3 Prínos metodiky

Ak je princíp aplikovaný na menšie logické celky, ktoré nie sú nevyhnutne potrebné k pochopeniu hlavného toku procesu, ich "skrytím" a zoskúpením pod komponentu subprocesu výrazne zvyšujeme jeho prehľadnosť a čitateľnosť.

V prípade potreby získania detailnejšej informácie o fungovaní danej časti je možné subproces jednoducho otvoriť a pracovať tak ako v prípade jeho viditeľnosti v rámci hlavného procesu.



Obr. 6.7: Výsledný stav zjednodušeného procesu

6.1.2.4 Zhrnutie vo všeobecnosti

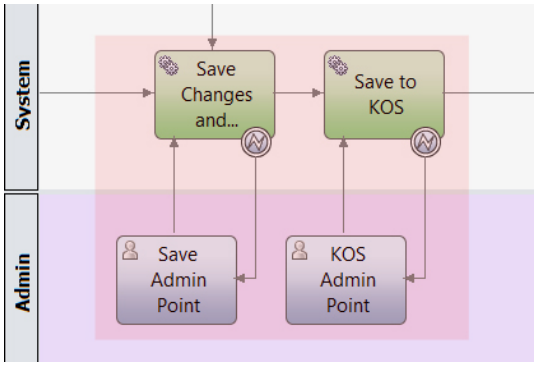
Zmena aplikovateľná na akékoľvek logické celky procesu, ktoré priamo nesúvisia s jeho celkovým vnímaním a zároveň sú medzi sebou logicky previazané. Ich význam je zahrnutý formou súhrnného pomenovania komponenty subprocesu, takže v prípade potreby sú jednoducho dohľadateľné.

6.1.3 Servisné volania

Ide o bežne využívanú komponentu, ak potrebujeme získať alebo spracovať dáta v rámci procesu.

6.1.3.1 Problém

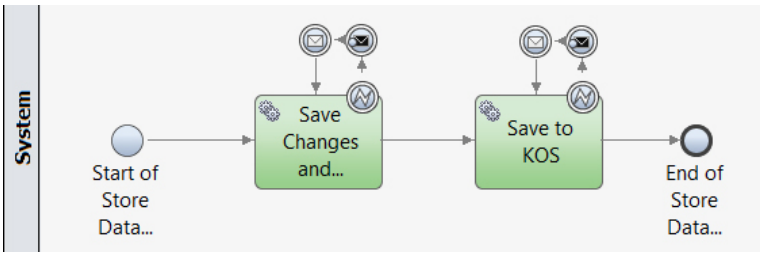
Častý problém je, že týchto volaní je zbytočne mnoho a týmto znižujú prehľad a orientáciu v rámci celého procesu. 6.8



Obr. 6.8: Dve servisné komponenty za sebou a navyše bez “koloběžky“

6.1.3.2 Riešenie

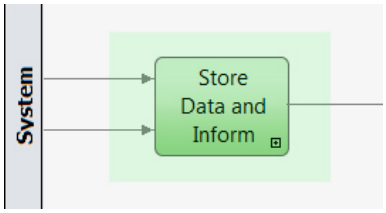
Pri riešení si položíme otázku, či je možné nahradzovať alebo iba zoskupovať. V prípade, že komponenty riešia príbuzné veci a ich využitie sa v iných procesoch neopakuje, je na nás ku ktorej z možností sa prikloníme. Spravidla sa vytorí jedna väčšia vrstva a proces sa zjednoduší. V opačnom prípade, ak komponenty riešia dve nezávislé veci je preferovaná možnosť zoskupovania pod subproces. Ďalej ale nastáva otázka, či je možné naviazať error handling rovno na celú komponentu subprocesu. V tomto prípade nie, pretože model jasne vyžaduje odchytávanie chybových stavov na konkrétnej komponente 6.9, na ktorej nastal a teda naviazaním o "vrstvu vyššie"by nebolo hneď jasné v ktorej z dvoch servisných komponent chyba nastala.



Obr. 6.9: Výsledná podoba servisných komponent presunutá do samostatného subprocesu

6.1.3.3 Aplikácia metodiky

Výsledkom je znova súhrnne pomenovaný subproces, ktorý v sebe zoskupuje ukladanie dát a notifikáciu.6.10



Obr. 6.10: Finálna podoba časti procesu formou subprocesnej komponenty

6.1.3.4 Zhrnutie vo všeobecnosti

Otázka sa môže klásť pri akejkoľvek situácii, ktorá obsahuje viacero servisných komponent, no až odpovede na nich nám naznačia do akej miery je možné túto metódu použiť a či vôbec.

6.1.4 Poradie premenných

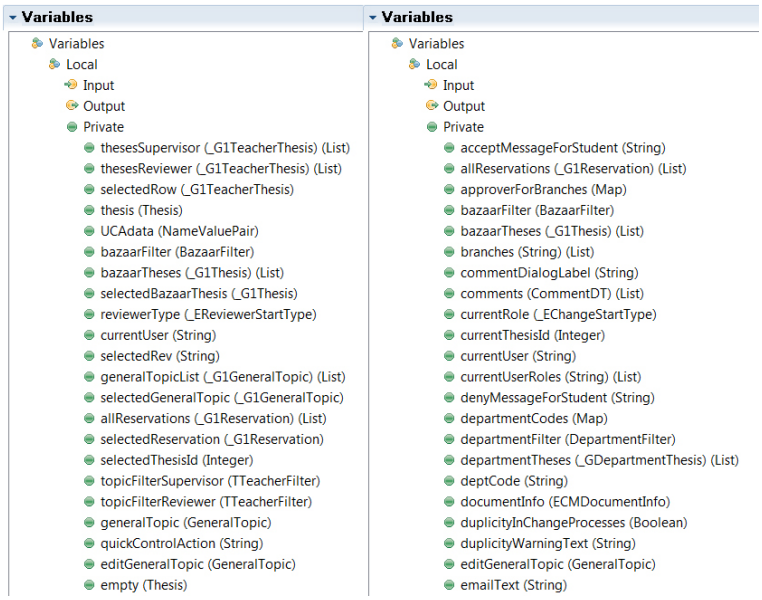
Na prvý pohľad banálna záležitosť vnímaná ako samozrejmosť, no prax ukazuje ako často sa táto konvencia nedodržiava. Tento úkon nemá žiaden vplyv na výkon či rýchlosť aplikácie, no značne ovplyvňuje efektivitu vývoja či znižuje riziko zanesenia chyby a to v prípade, že je potrebné hľadať niektorú z premenných, ktorej výskyt je skrytý medzi ďalšími objektami či dvadsiatkou iných premenných.

6.1.4.1 Problém

V aplikácii ZP sa nezoradený zoznam premenných 6.11 nachádzal prakticky všade, kde sa dajú dejinovať premenné a teda celkové dohľadávanie bolo časovo náročné.

6.1.4.2 Riešenie

Riešenie je jednoduché, zoradiť. Bežným radiacim kritériom je abeceda s tým, že premenné sú občas pomenované i prefixami na základe čoho je možné i radenie do skupín podľa príslušnosti, kde je ale na zváženie využitie objektov združujúcich práve takéto skupiny atribútov.



Obr. 6.11: Zoznam premenný pred a po zoradení podľa kritéria abecedy

6.1.4.3 Aplikácia metodiky

Zoradený zoznam dáva lepší prehľad o tom, či daný atribut existuje, prípadne už nie je pridaný podobný, splňujúci rovnaký účel.

6.1.4.4 Zhrnutie vo všeobecnosti

Do probému riešenom v tejto časti by sa vyvíjaná aplikácia nikdy nemala dostať, preto je nevyhnutné si tento návyk osvojiť a striktne dodržiavať.

6.2 Frontendová vrstva

Optimalizácie na úrovni fontendovej vrstvy v sebe nesú úpravy v rozložení jednotlivých častí obrazoviek, kde sú volané alebo použité. Táto vrstva má teda na svedomí zdĺhavé (príp. viacnásobné) vykresľovanie jednotlivých komponent či nekonečné prechody medzi obrazovkami.

6.2.1 Členenie na Coach Views

Ďalším neodmysliteľným krokom k dosiahnutiu dobre udržiavaného a prehľadného kódu je členenie jednotlivých frontendových častí do znovupoužiteľných Coach Views. Princíp spočíva najprv v indetifikácii komponenty, ktorá je používaná na viacerých miestach. Pre tento účel sa odporúča vytvorenie komplexného pohľadu formou diagramu kde sa dajú opakované miesta jednoduch-

šie vyhľadávať. Takto identifikované celky vytvorím ako znovupoužiteľné CV, ktoré v sebe budú zapúzdrovať všetku potrebnú funckionalitu a reagovať na zmeny na základe vstupných parametrov ako je napríklad viditeľnosť tlačidiel či volanie iných služieb v závislosti na tom, pod akou rolou je daný CV používaný. Tento princíp v sebe nesie zmysel samotného DI, definovanom v sekcii o navrhových vzoroch. Ja si v CV definujem parametre, ktoré potrebujem a ten, kto ho potrebuje použiť musí zabezpečiť aby boli korektne predané. Vytvorený CV zapojím tak, že parametre vstupujúce do pôvodných komponent predám cez binding a konfiguračné nastavenia novovzniknutému CV.

Aplikácia záverečných prác spomínané členenie na CV obsahuje len zriedkakedy čo značne komplikuje vyoknávanie akejkoľvek dodatočnej zmeny v aplikácii bez zvýšeného rizika zanesenia chýb.

Proces aplikácie tejto zmeny bol pre mňa, ako človeka bez hlbších implementačných znalostí v rámci tohto projektu, pomerne komplikovaný. Nemal som dostatočné znalosti o fungovaní aplikácie ako celku, či vzájomných závislostí jednotlivých častí. Preto som prechodom všetkými modulmi vytvoril screenshoty obrazoviek, predstava akéhosi diagramu, a takéto miesta označil. Pre lepšiu predstavu je diagram dostupný v prílohe D. Tieto obrazce som zoradzoval tak ako po sebe následovali v aplikácii, následne zoskupoval podľa Human Services a napokon ohraničil a označil ako jeden modul. Na takomto komplexnom pohľade som postupne nachádzal podobné či rovnaké časti, pridával im tagy, a podľa toho vytvoril Coach view, ktorým som mohol všetky tieto časti nahradiť. CVs som podľa výskytov triedil do kategórii TK alebo vlastný.

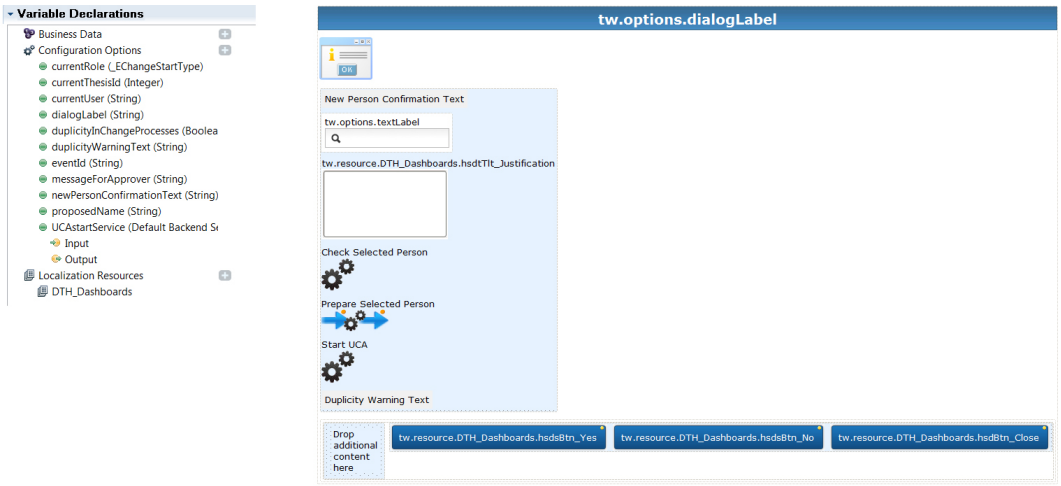
TK CVs boli také, ktoré mali výskyt i mimo vlastnú aplikáciu. Tieto typy som definoval, ako z označenia vyplýva, v Tookitoch ktoré boli všobecne použité vo všetkých moduloch. Bolo to z dôvodu, že takéto komponenty boli využívané vo viacerých moduloch, preto by bolo potrebné ich definovať pre každý modul zvlášť. Takto postačilo iba volanie prostredníctvom Toolkitu, ktorý bol pre všetky moduly spoločný. V prípade, že šlo o vlastné, a teda ich výskyt iba v rámci príslušného modulu, bolo postačujúce definovanie iba v tom danom module.

6.2.1.1 Problém

Kód dialogového okna, ktoré vidíte na obrázku 6.12 sa v rámci aplikácie nachádzal tri krát. V prípade zmien, či rozvrhnutia prvkov bolo potrebné upravovať každú časť zvlášť, čo zvyšovalo šancu zanesenia chyby pri úpravách.

6.2.1.2 Riešenie

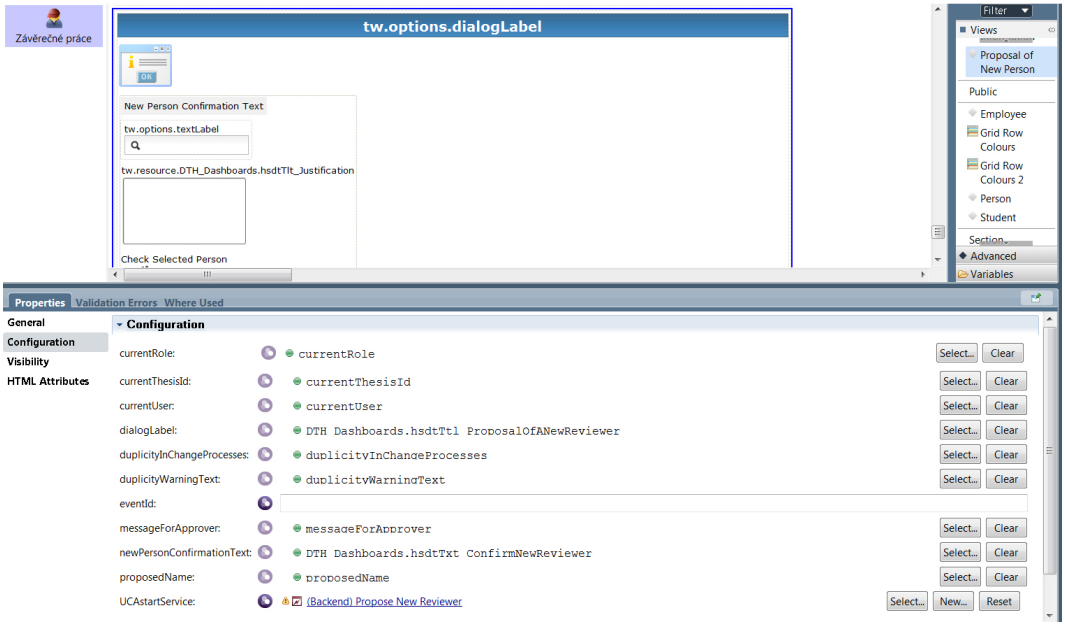
Vytvorené Coach View, ktoré vidíte na obrázku v sebe nesie všetky časti pôvodného okna a navyše je parametrizované tak, že sa dá použiť všeobecne pre všetky tri prípady.



Obr. 6.12: CV konštruované na základe niekoľkonásobne používanej časti

6.2.1.3 Aplikácia metodiky

Na následujúcom obrázku 6.13 je viditeľné použitie znovupoužiteľného CV, ktoré sa síce na Human Servise nachádza stále tri krát, no vždy sú mu predávané iné parametre v závislosti na tom, pre akú rolu sa má vykresliť. Teda v tomto prípade ide o premennú `currentRole`, ktorá v sebe nesie príslušnú rolu, na základe ktorej sa v CV zobrazujú príslušné dáta a tlačidlá. Za povšimnutie stojí i parameter `UCAstartService`, ktorého vstupom je rovno služba, ktorá zabezpečuje otvorenie korektného okna či odoslanie dát na spracovanie službou pripravenou pre danú rolu. Všetky parametre sú predávané v záložke "configurations" formou ako je demonštrované na obrázku 6.13.



Obr. 6.13: Zapojenie a predanie vstupných parametrov vytvoreného CV

6.2.1.4 Zhrnutie vo všeobecnosti

Táto operácia slúži len ako názorný príklad vytvorenia znovupoužiteľného CV. Existuje nekonečne mnoho možností kde sa dá tento princíp aplikovať, no podstatné je uvedomiť si, že takáto možnosť exisuje a je potrebné ju používať, čim sa nám výrazne zjednodušuje rozšíriteľnosť či udržateľnosť jednotlivých komponent aplikácie.

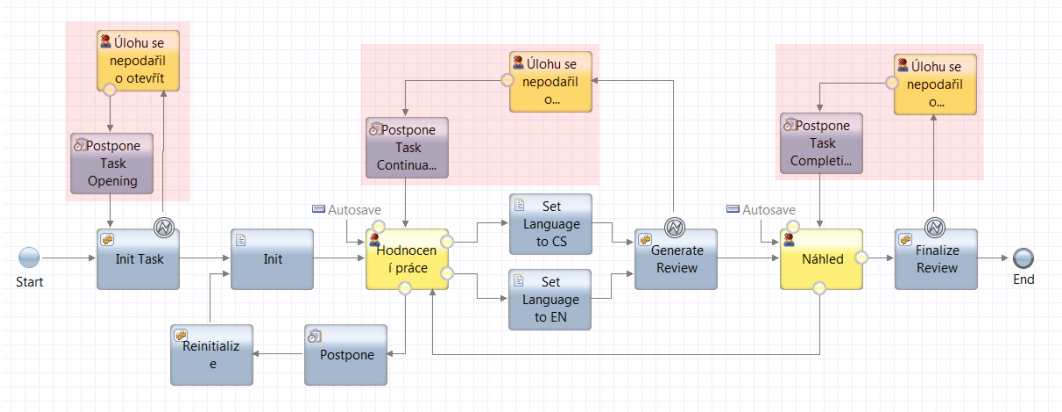
6.2.2 Členenie kódu do Toolkitov

Táto rada môže byť chápaná ako nadstavba na predchádzajúcu optimalizáciu. A teda ak máme screeny rozdelené do menších Coach Views, je dobrým zvykom, v prípade použitia naprieč viacerými aplikáciami, takýmto častiam vytvoriť vlastný Toolkit. Je dôležité si uvedomiť, že tak ako sú znovupoužiteľné Coach Views, to isté platí i pre služby, objekty či iné komponenty IBM BPM.

6.2.2.1 Problém

Ako demonštratívny príklad bol v aplikácii záverečné práce nájdený celok 6.14, ktorý bol opätovne implementovaný 14 krát. Je pochopiteľné, že vykonať akú-

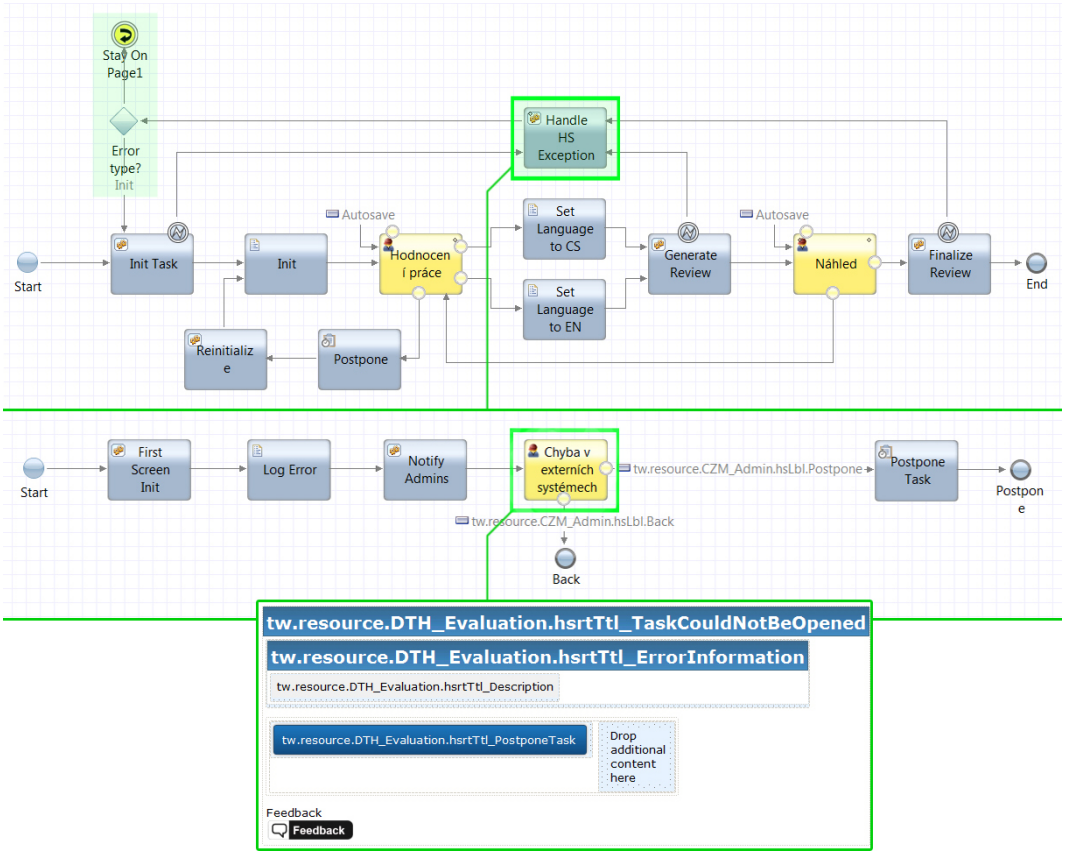
koľvek zmenu, naprieč všetkými jeho použitiami vo všetkych aplikáciach je časovo náročné s vysokou šancou zanesenia chyby.



Obr. 6.14: Flow zapojených Coaches obsahujúcich chybové hlášky

6.2.2.2 Riešenie

Implementácia univerzálnej Human Service obsahujúcej Coach View, ktorá je parametrizovaná a zapúzdruje všetku logiku je v tomto prípade žiadúca. Je vhodné taktiež aplikovať pravidlo, že ak je používaná naprieč viacerými aplikáciami, jej implementáciu vložíme do Toolkitu. Jej zapojenie 6.15 je demonštrované na následujúcom obrázku.



Obr. 6.15: Centralizované zapojenie Human Servisy nesúcej funkcionalitu pôvodného Coacha, služba z vnútra, náhľad Coach View

6.2.2.3 Aplikácia metodiky

Rozdelením podobných komponent do CV a niektorých i do TK sa programová časť aplikácie stala prehľadná, omnoho ľahšie rozšíriteľná a hlavne znovupoužiteľná. V prípade zmien v týchto komponentách nie je potrebné zmeny aplikovať niekoľkonásobne naprieč všetkými použitiami a teda zbytočne zvyšovať riziko chyby.

6.2.2.4 Zhrnutie vo všeobecnosti

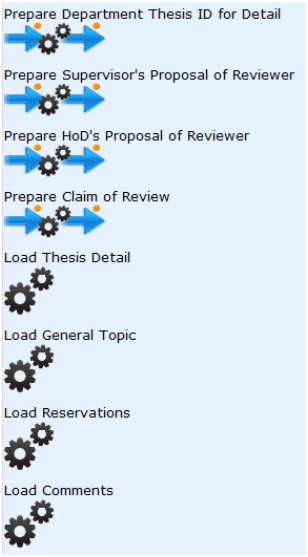
Vo všeobecnosti platí, že vždy je lepšie každú komponentu vytvárať ako CV, pretože nikdy nevieme, kedy sa nám v budúcnosti môže hodiť. Ak nebude zodpovedať naším požiadavkom, tak sa jednoducho vytvorí nová a nič sa tým nestratí.

6.2.3 Volanie AJAXových služieb

Pri začiatkoch vývoja aplikácie ZP technológia neumožňovala iné typy volaní, takže v prípade handlingu rôznych požiadaviek bolo potrebné vytvoriť väčšie množstvo naslúchajúcich "Intermediate eventov".

6.2.3.1 Problém

Pri spúšťaní jednotlivých CV, BPM potrebuje inicializovať všetky udalosti, ktoré sa môžu niekedy počas behu daného CV volať. Teda v tomto prípade so spustením obrazovky inicializuje osem slotov 6.16, ktoré sú pripravené reagovať na nejakú požiadavku. Tieto operácie sú drahé z pohľadu pamäte, ktorá je zbytočne zaťažovaná a nie je vôbec isté, či inicializované udalosti budú vôbec zavolané.



Obr. 6.16: Súbor inicializovaných udalostí

6.2.3.2 Riešenie

Elegantné riešenie prišlo až časom a to implementáciou funkcionality zvanej script flow, kde je vložená komponenta v sebe schopná niesť, resp. zapúzdrovať súbor udalostí, ktoré sú vykonávané ďalej na základe určitých pravidiel, pričom v pamäti vystupuje ako jediná udalosť a teda zaberie jediný slot.

6.2.3.3 Aplikácia metodiky

Aplikácie navrhnutého postupu výrazne znižuje pamäťové nároky aplikácie, čo má taktiež značný vplyv na výkon a odozvu.

6.2.3.4 Zhrnutie vo všeobecnosti

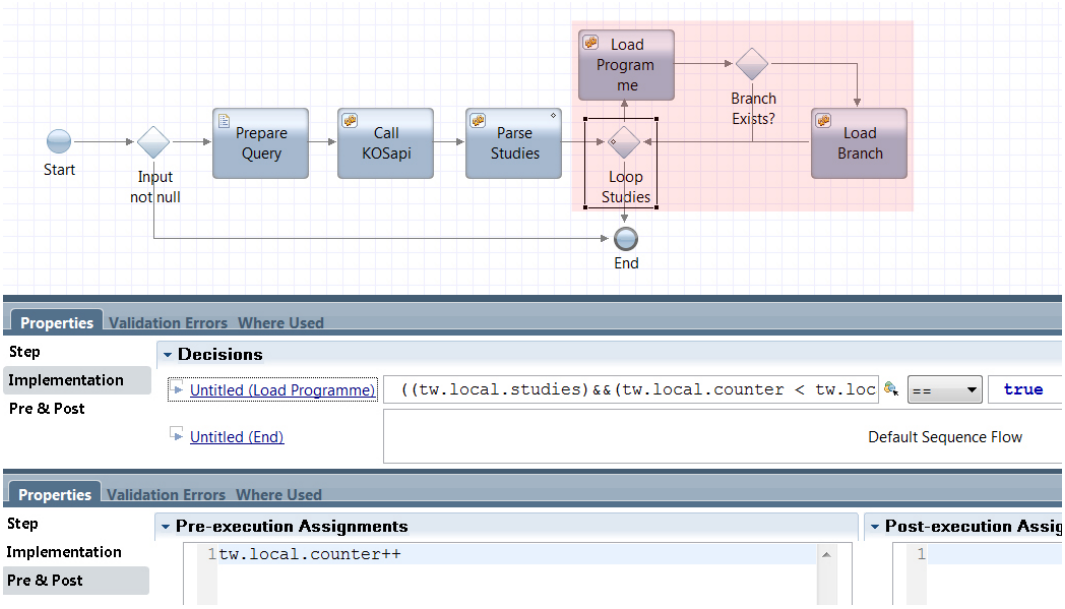
Spomínaná metóda je aplikovateľná len v prípade, že v danej aplikácií je používaný Coach TK verzie 6.5 a vyššej, pretože od tejto verzie tam existuje príslušná komponenta Script Flow.

6.3 Servisná vrstva

Optimalizácie tejto časti považujem za najdôležitejšie pretože ide o rýchlosť, odozvu a celkovú flexibilitu pri práci s aplikáciou, ktorú ocenia nie len užívatelia, testeri ale častokrát i samotní vývojári. Ide o backendovú časť, kde bolo cieľom zlepšenie služieb a to najmä odstránením nadbytočných volaní a requestov do databáze, či zlepšiť ich škálovateľnosť z hľadiska znovupoužiteľnosti. Tieto postupy a úpravy som zdokumentoval, aby boli uchopiteľné a použiteľné pri ktorejkoľvek aplikácii v IBPM.

6.3.1 Viacnásobné volania služby

Väčšina novovytvorených jednoduchých služieb slúži iba k vráteniu záznamu na základe vstupnej hodnoty. V prípade, že je potrebné získať hodnôt viac nastáva problém. Rýchlym riešením, bez potreby upravovať službu je vytvorenie cyklu formou gatewaye, ako môžete vidieť na obrázku 6.17



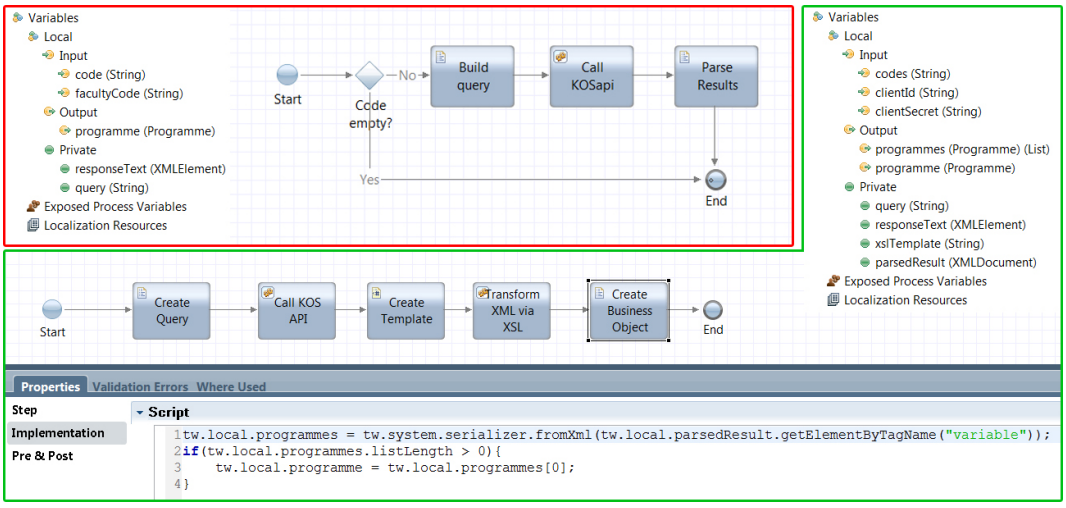
Obr. 6.17: Zapojenie, v ktorom je v cykle niekoľkokrát volaná rovnaká služba

6.3.1.1 Problém

Táto transformácia je v prípade veľkého množstva iterácií mimoriadne neefektívna, pretože každý prechod cyklom znamená, že zlužba je znovu volaná, len z iným parametrom a spravidla je inkrementovaný hneď v "post" záložke komponenty rozcestníka. Hovorovo ide o "for" cyklus implementovaný v BPMN. Súbor takýchto operácií dosť výrazne znižuje odozvu aplikácie a spomaľuje chod systému.

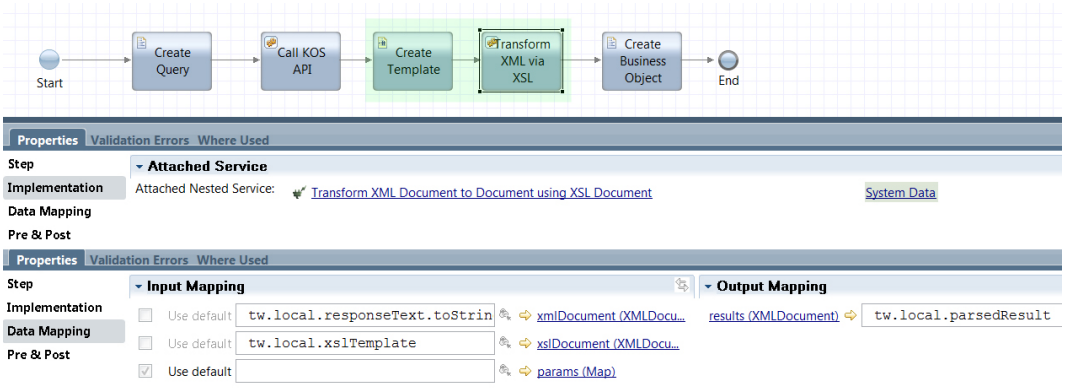
6.3.1.2 Riešenie

Pri riešení takéhoto problému je nevyhnutné službu poupraviť a optimalizovať tak, že bude schopná na vstupe spracovať i kolekciu, príp. reťazec - string, a teda zoznam uložený v jednom zo vstupných parametrov 6.18. Znamená to, že službu zavolám iba raz, ale dostanem kompletný výsledok ako za bežných okolností pri uplynutí všetkých iterácií cyklu. Vrátenú kolekciu dát spracujem úplne rovnako a používam tak ako doteraz. Pre zaujímavosť pripomeniem, že táto služba je implementovaná podľa návrhového vzoru Adaptér, ktorý som predstavil v kapitole o frameworkoch. Vzor je aplikovaný formou spracovávania vsupu, v prvej skriptovej komponente, volanie externej služby a v ďalších komponentách spracovávanie dát poskytnutých službou. V prípade, že sa zmení externá slúžba na inú, v našom prípade KOS API, servisnú komponentu nahradíme a všetky zmeny vykonáme úpravou ostatných komponent, ale vždy len v rámci tejto služby, pretože systém s ňou počíta a je odstienený od jej vnútornej implementácie.

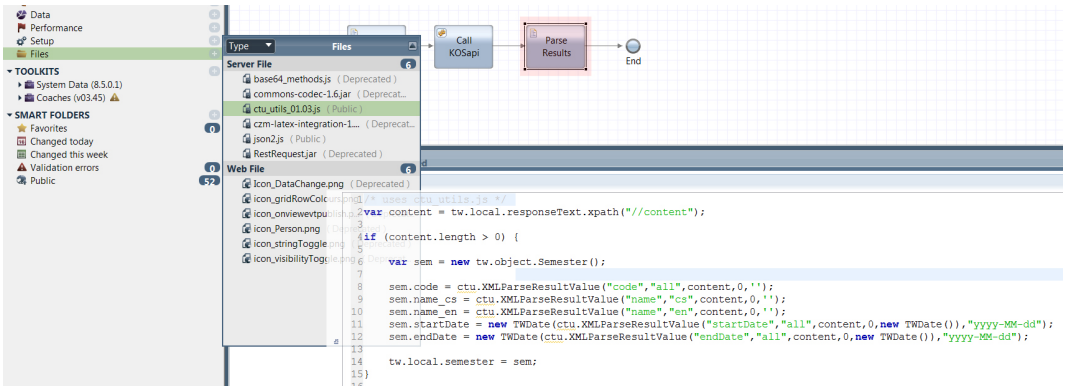


Obr. 6.18: Obsah a premenné pôvodnej a novej služby

Zobrazenie korektného tela služby môže pôsobiť mätúco, netreba v tom hľadať zložitosť. Prvá "server side script" komponent z obrázka 6.18 sa rozšírila o spracovávanie rôznych vstupov, kde pred tým spracovávala iba jeden prvok, teraz ich v stringu parsuje a prijíma niekoľko. Na základe nich potom konštruuje query, ktorý volá v druhej, servisnej komponente, ktorá ostala nemenná. Teraz nastáva najväčšia zmena a to v spôsobe získavania dát. Obsah pôvodného parsovania je viditeľný na obrázku 6.20, kde sú používané JS funkcie z iného súboru. Pre neznalých je táto operácia v poriadku, je ale dobre dávať prednosť predpripraveným funkciám, ktoré nám poskytujú Toolkity ako je v našom prípade TK s názvom "System Data". Ten v sebe nesie transformáciu XSL šablóny do business objektu, ktorý je ďalej používaný samotným BPM. Operácie cez externý Javascript majú sčasti vplyv na pamäť, no podstatnejšie je, že skompilovaný kód, ktorý danú transformáciu vykonáva sa môže kedykoľvek za behu z pamäte uvoľniť. Je preto zbytočné siahať nie pre niečo, čo už máme dávno k dispozícii, a zbytočne zvyšovať riziko zanesenia chyby. Spomenuté operácie sa týkajú vyznačených komponent, viď obrázok 6.19. Vzor XSL šablóny je zverejnený v prílohe C.



Obr. 6.19: Zapojenie využívajúce funkcionalitu System Data TK



Obr. 6.20: Pôvodné zapojenie využíva funkcie z externého súboru

6.3.1.3 Aplikácia metodiky

Táto zmena priniesla ďalšiu, dobre optimalizovanú službu, ktorá výrazne znížila dobu odozvy a spracovávania informácií. Táto skutočnosť vyplýva z meraní a zvýšenia rýchlostí v rámci všetkých miest kde je použítá.

6.3.1.4 Zhrnutie vo všeobecnosti

Demonštrované úpravy sa javia ako samozrejmosť, no je potrebné na to myslieť a nenechať sa uniesť jednoduchosťou a pocitom, že niečo konečne funguje. V prípade služieb to platí obzvlášť, keďže ide o základný stavebný pilier každej aplikácie. Ak sú služby navrhnuté a optimalizované zle, chyba sa dedí a nabaľuje, keďže sú väčšinou používané opakovane.

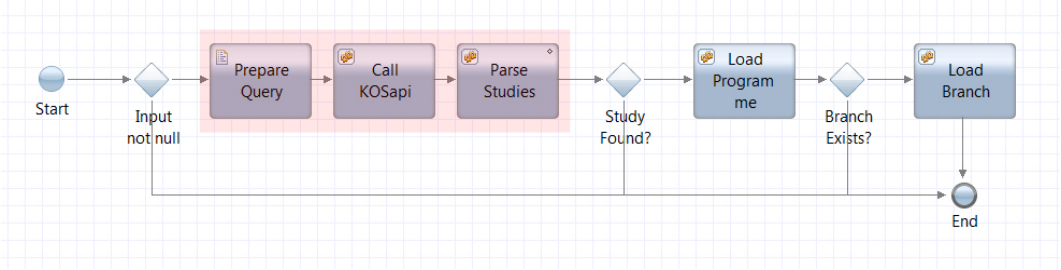
6.3.2 Znovupoužiteľnosť služieb

Pri vývoji je dôležité implementovať funkcie či služby tak, aby ich bolo možné znovu používať aj v inom kontexte. Zabezpečiť to ale nie je jednoduché a to najmä v prípadoch, kedy nie je vopred zrejmé na čo všetko bude daná služba používaná. Vo všeobecnosti platí pravidlo, že čím je rozsah pôsobenia služby väčší a teda čím viac problémov sa snaží riešiť, tým klesá jej znovupoužiteľnosť. Rozhodnutie ktorým smerom sa má uberať je teda vždy v rukách vývojára.

6.3.2.1 Problém

V aplikácii ZP sa častokrát vyskytovali služby, ktoré riešili množstvo podobných problémov, no práve kvôli tomu bolo vytvorených i množstvo veľmi podobných a len v malej časti líšiacich sa služieb. Názorný príklad je demonštrovaný na obrázku 6.21, kde v jednej službe získavame študijné informácie o

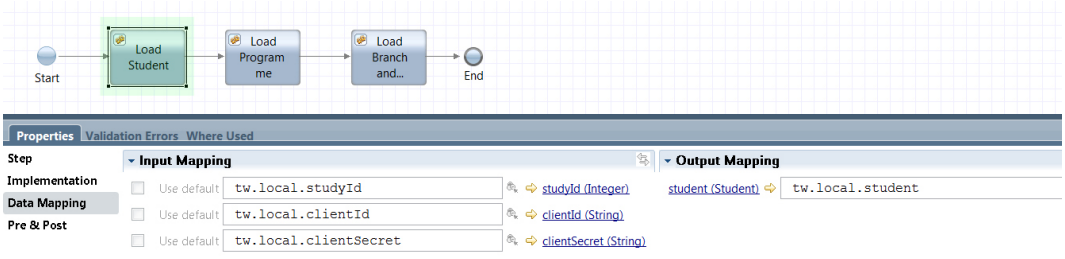
študentovi. Služba je zbytočne komplikovaná a vykonáva nadbytočné, časovo náročné operácie.



Obr. 6.21: Chybné zapojenie komponent v obsahu služby

6.3.2.2 Riešenie

Pri riešení takéhoto problému je dôležité uvedomiť si, či novovzniknutá služba bude mať zmysel, prípadne čo konkrétne bude riešiť. To nám dá predstavu, ako by mala vyzerať z pohľadu parametrov a obsahu. Z architektonického pohľadu je zrejmé, že práca so študentami a teda získavanie k nemu patriacich informácií je pomerne frekventovaný úkon a teda je to vhodný adept na samostatnú službu, ktorá sa môže prípadne využívať v rámci iných, zložitejších služieb. Znova pripomeniem princíp DI, ktorý uplatníme pri definovaní služby, ktorá plní určitú funkcionalitu. Ak ju potrebujem, službu zavolám a zabezpečím aby pracovala s korektnými vstupnými dátami. Mapovanie vstupných a výstupných parametrov takejto služby je zachytené na obrázku 6.22.



Obr. 6.22: Správne zapojenie komponent v obsahu služby s náhľadom mapovacích parametrov

6.3.2.3 Aplikácia metodiky

Po úpravách získavame prehľadnú službu obsahujúcu ďalšie tri, ktoré už nie sú prepojené cez rozcestníky – vetvenia, ako je viditeľné na obrázku 6.22, a rozličné iné volania. Všetko potrebné je v nich zapúzdrené a teda ten, kto ich používa nemusí riešiť prípady "čo ak". Služba by mala fungovať ako samostaný celok, ktorý na základe vstupu poskytuje adekvátny výstup.

6.3.2.4 Zhrnutie vo všeobecnosti

Zmenou získame lepší prehľad pri skladaní komplexnejších služieb, kde je hneď jasné a väčšinou z názvu komponenty vyplývajúce čo daná služba poskytuje. Táto problematika je úzko spätá i s rozhodnutím, kedy je lepšie v službe jedným volaním vrátiť rozsiahly objekt, ktorý sa následne parsuje a rozkladá na menšie časti v rámci scriptov, ako pre každú časť volať tú istú službu len s rozdielnymi parametrami.

6.3.3 Uvoľňovanie premenných na konci služby

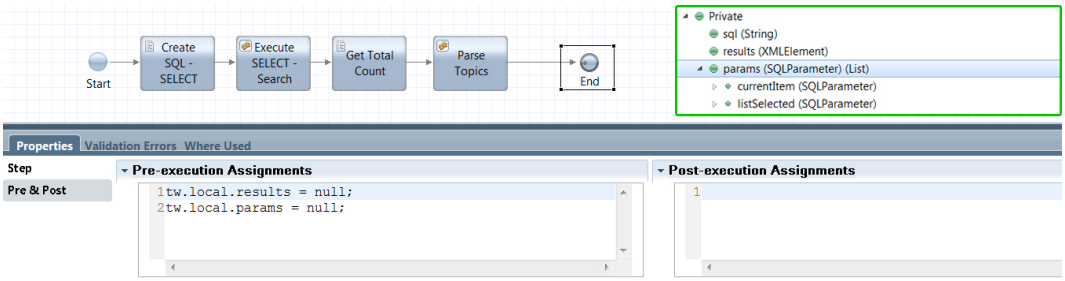
Práca s veľkými objektami je vo väčších aplikáciách štandardom. V prípade IBM BPM ale nastáva pri službách menší problém, ktorý navonok nie je viditeľný.

6.3.3.1 Problém

Ide o uvoľňovanie pamäte, konkrétne premenných, aby sa zbytočne neukladali a nezaťažovali tak BPM databázu. V prípade menších projektov je táto skutočnosť zanedbateľná, ale pri projektoch väčších rozmerov, ako i SZP, sa výskyty takéhoto typu opakuju častejšie a preto čas odozvy zbytočne narastá.

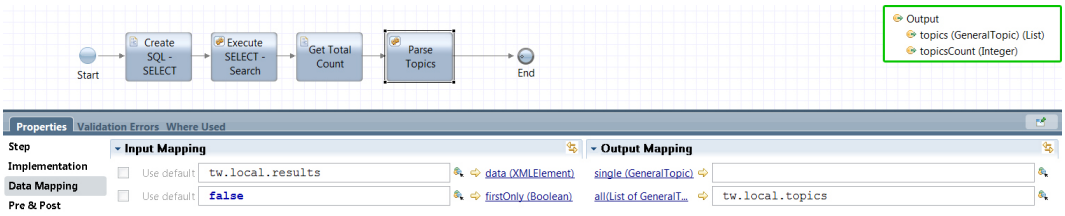
6.3.3.2 Riešenie

Rišenie je v princípe jednoduché. Stačí na konci služby "vynullovať" všetky parametre, prípadne celý objekt postupne, premennú po premennej tak ako to býva u inicializácií. Táto operácia sa odporúča vykonávať pri záložke "Pre-execution" v "End" komponente, viď obrázok 6.23.



Obr. 6.23: Ukážka "nullovania" objektov a ich štruktúra

Dáta, ktoré sa mapovali do služby ako parameter boli spracované, predané do výstupnej premennej 6.24 a preto "uvoľnenie pamäte" nijak neovplyvní návratové hodnoty služby.



Obr. 6.24: Presun dát (mapovanie) na výstupnú premennú

6.3.3.3 Aplikácia metodiky

Tak ako bolo spomenuté, táto operácia prináša výrazné zlepšenie doby odozvy samotnej aplikácie, čo je pre užívateľov a ich pohodlnosť pri práci so systémom kľúčové.

6.3.3.4 Zhrnutie vo všeobecnosti

Údaje z Human Service nie sú uvoľňované až do okamihu, kedy služba dosiahne konečný bod (endpoint). V článku [18] sa píše, že ak je služba implementovaná, bez predpokladu dosiahnutia takéhoto bodu, ako je napríklad samostatná stránka či presmerovanie, pamäť nebude uvoľnená až po dobu timetoutu, čo je spravidla až 2 hodiny. Preto je potrebné vždy v situáciách spĺňujúcich daný popis tento stav ošetrovať.

6.4 Zhrnutie kapitoly

V tejto kapitole som formuloval metodiku, ktorej použitie som následne demonštroval na praktickom príklade systému ZP. Do akej miery je metodika prínosná v rámci systému ZP je zhrnuté v následujúcej kapitole. V závere musím konštatovať, že aplikáciou metodiky na akýkoľvek zvolený systém postavený na technológii IBPM, je veľmi pravdepodobné dosiahnutie zlepšenia niektorých kvalitatívnych parametrov a to z pohľadu vývoja, využiteľnosti a údržby (celá dimenzia procesu životného cyklu), infraštruktúry (dimenzia vlastností) a použiteľnosti (dimenzia kvalitatívnych charakteristík) viď 7.1. Zlepšenie ostatných vlastnosti ako je funckionalita, obsah či spoľahlivosť aplikáciou metodiky nie je zaručená. Tieto fakty sú v následujúcej kapitole podložené výsledkami testov vykonávaných na aplikácií ZP.

Kapitola 7

Zhodnotenie a testovanie

Bežnou praxou pri vývoji väčšieho systému je retest celej aplikácie skúsenejším týmom testerov. Platí samozrejme, že každá novo implementovaná funkcionalita má byť zároveň testovaná samotným vývojárom, no občas sa môže na niečo zabudnúť, prípadne stratiť všeobecný prehľad nad tým, ktoré časti, ktoré pred tým fungovali, mohli byť ovplyvnené. Väčšinou sa testuje vždy po implementovaní menšieho funkčného celku, no v tomto prípade sa celé testovanie odložilo na koniec, teda po aplikácií všetkých optimalizačných pravidiel, a vykonalo sa naraz. Testovalo sa z dvoch pohľadov a to užívateľského a vývojarského. Ten druhý sa pokúsim rozobrať v následujúcich riadkoch.

7.1 Typy frontendových testov

Testovaníe behu aplikácie, dostupnosti častí procesov a funcionalít prebiehalo tradične formou scenárov. Keďže aplikácia má prvý release za sebou, nebolo potrebné vyhotovovať nové ani obmieňať stávajúce, keďže sa žiadna nová funkcionalita neimplementovala. Testy odhalili pomerne veľa bugov, ktoré boli i vďaka optimalizácii rýchlo a efektívne odstránené. V princípe sa testoval priamočiary proces, ktorý sa vetví len zdriedka. Prebieha od vytvorenia zadania práce, cez schvaľovací proces až po akceptáciu, nahranie zo strany študenta, oponentúru a napokon vyhodnotenie.

7.2 Typy vývojárských testov

Tieto testy merali hlavne odozvu a rýchlosť načítania jednotlivých častí aplikácie.

Načítavanie jednotlivých stránok sa výrazne zlepšilo a to najmä zavedením lazy loadingu. Ide o metódu, kde sa nenačítava celý obsah naraz, ale len časti -komponenty, ktoré sa nachádzajú v aktuálnom viewporte užívateľa. Metóda bola aplikovaná i formou editovateľnosti textových polí, kde komponenta

na začiatku vie, že nebude editovateľná, takže sa neinicializuje všetok javascript, ale len ten, ktorý je nevyhnutný pre prácu v readonly režime. Vplyvom týchto úprav klesá celková réžia a množstvo dát potrebných pre viditeľnú či editovateľnú časť stránky, teda obsah je užívateľovi zobrazený skôr.

Znížila sa i celková pamäťová náročnosť, čo bolo dosiahnuté skracovaním timeoutu v KOS API. Šlo o úpravu, kde bola nastavená kratšia doba udržiavania dát v cache pamäti a teda nutnosť uchovávať veľké datové objekty. Týmto vplyvom ale o niečo málo vzrástla doba odozvy, keďže je potrebné server žiadať o zaslanie aktuálnych dát častejšie.

Z hľadiska rýchlosti boli zlepšené i časti, v ktorých sa pracuje so súbormi a to najmä nahrávanie. Spracovávanie bolo nahradené formou Latexu, ktorý je tak ako docx, síce spracovávaný Javou, no pri Latexe využíva operácie OS, ktoré s dátami pracujú efektívnejšie.

7.3 Merania a testy rozšíriteľnosti

Druhá časť priniesla, podľa očakávaní, dobré výsledky a teda optimalizáciou bolo dosiahnuté zlepšenie najmä v oblasti rozšíriteľnosti a rýchlosti. Rozšíriteľnosť bola meraná analyticky a to zadaním požiadavku rozšírenia, ktorého doba realizácie bola diskutovaná v stavoch pred a po. Ako príklad slúži zmena rozloženia prvkov komponenty informujúcej o chybnom či nesprávnom postupe užívateľa. Pôvodne by zmena tohoto požiadavku trvala odhadom 6 hodín, no vďaka aplikovaniu pravidla univerzálnosti a znovupoužiteľnosti komponenty bola doba úpravy stkrátená a odhadnutá na 1 hodinu.

Časové úspory v rámci vykreslenia požadovanej obrazovky boli merané naprieč celou aplikáciou o čom vypovedajú následujúce tabuľky. Výsledné časy sú uvedené v sekundách.

Tabuľka 7.2: Priradenie oponenta ZP

Špecifiká pohľadu		**Rýchlosť zobrazenia**	
Rola	**Názov pohľadu**	**Pred**	**Po**
Budúci op. ZP	Prejavenie záujmu o oponentúru ZP v burze voľných oponentúr	5.371	3.225
Kat. schval.	Navrhnutie oponenta	3.974	3.024
	Schválenie oponenta	2.024	1.961
Vedúci ZP	Vytvorenie rámcovej TZP	3.812	2.912
	Navrhnutie oponenta	3.533	3.342
Oponent ZP	Vyjadrenie súhlasu s oponentúrou	2.045	2.037

Tabuľka 7.1: Vytvorenie rezervácie a schvaľovania ZP

Špecifiká pohľadu		**Rýchlosť zobrazenia**	
Rola	**Názov pohľadu**	**Pred**	**Po**
Vedúci	Vytvorenie rámcovej TZP	6.257	4.134
	Vytvorenie rezervácie a schvaľovanie ZP	4.311	3.802
	Schválenie rezervácie študenta	2.154	2.158
Študent	Prejavenie záujmu o obor	2.076	1.891
	Záväzná voľba rámcovej témy	2.053	1.785
	Schválenie konkrétneho zadania ZP	2.034	1.788
	Schválenie úprav v zadaní ZP	2.194	2.050
	Finálne prijatie zadania ZP	2.039	1.881
Vedúci ZP	Vytvorenie konkrétneho zadania ZP	3.854	3.547
	Prepracovanie zadania ZP	2.652	2.044
	Finálne prijatie zadania ZP	2.084	1.921
Kat. schval.	Schválenie zadania ZP	1.982	1.995

Tabuľka 7.3: Vloženie a odovzdanie vypracovanej ZP

Špecifiká pohľadu		**Rýchlosť zobrazenia**	
Rola	**Názov pohľadu**	**Pred**	**Po**
Študent	Vloženíe a odovzdanie vypracovanej ZP vo formáte PDF	9.214	7.807
	Rozhodnutie o eskalácii neakceptovanej ZP	2.234	2.241
Vedúci ZP	Akceptácia vypracovanej ZP	1.894	1.731

Tabuľka 7.4: Vytvorenie a odovzdanie posudkov ZP

Špecifiká pohľadu		**Rýchlosť zobrazenia**	
Rola	**Názov pohľadu**	**Pred**	**Po**
Vedúci ZP	Zoznámenie sa s obsahom ZP	3.478	3.379
	Vytvorenie a odoslanie hodnotenia vedúceho ZP	8.254	6.035
Oponent ZP	Zoznámenie sa s obsahom ZP	4.479	4.269
	Vytvorenie a odoslanie posudku oponenta ZP	8.046	6.227

Tabuľka 7.5: Zmenové procesy v ZP

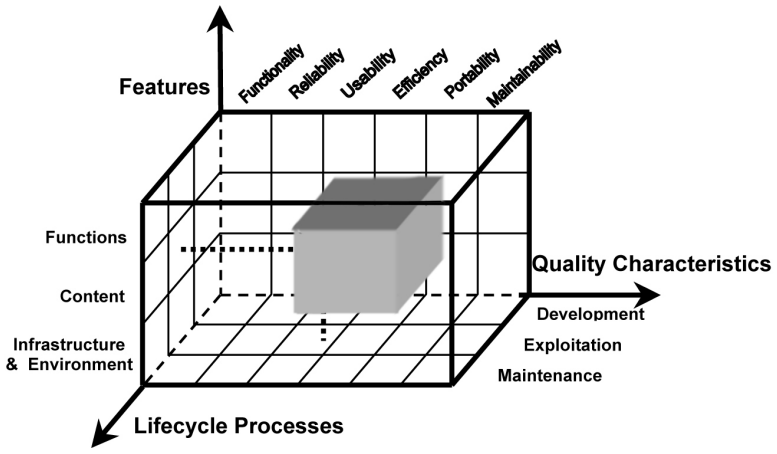
Špecifiká pohľadu		**Rýchlosť zobrazenia**	
Rola	**Názov pohľadu**	**Pred**	**Po**
Vedúci ZP	Návrh zmeny zadania ZP	3.053	3.072
	Špecifikácia zmeny zadania ZP	4.922	4.016
	Navrhnutie nového vedúceho	6.972	4.995
Kat. schvaľovateľ	Schválenie zmeny zadania ZP	2.043	2.055
	Navrhnutie nového vedúceho	5.726	4.039
	Schválenie vedoucího ZP	2.094	1.798
	Navrhnutie nového oponenta	6.102	5.392
	Schválenie oponenta	2.174	2.034
	Schválenie odzadania ZP	2.052	1.915
Študent	Schválenie zmeny zadania ZP	4.632	3.742
	Schválenie úprav v zadaní ZP	4.522	4.030
	Navrhnutie nového vedúceho	6.134	3.992
	Rozhodnutie o eskalácii návrhu na zmenu	2.421	2.326
	Žiadosť o odzadanie ZP	2.263	1.994
Fak. schvaľovateľ	Schválenie zmeny zadania ZP	5.886	5.131
Vedúci ZP	Prepracovanie návrhu zmeny zadania ZP	2.072	2.081
	Schválenie zmeny	3.011	3.053
	Navrhnutie nového oponenta	6.037	5.308
Nový vedúci ZP	Vyjadrenie súhlasu s vedením ZP	2.104	1.817
	Vyjadrenie súhlasu s oponentúrou	2.092	1.915
Oponent ZP	Navrhnutie nového oponenta	6.075	5.367

7.4 Zhodnotenie aplikácie metodiky na SZP

Optimalizovaním služieb sa zlepšil chod a odozva aplikácie o čom svedčia i testy v následujúcej kapitole. Aplikácia ostatných pravidiel nie je viditeľná pre bežných užívateľov aplikácie. Zmeny ale ocenia najmä vývojári, pre ktorých sa súčasný stav "kódu" aplikácie stal prehľadný, dobre škálovateľný a teda bez väčších problémov i ľahko rozšíriteľný.

Zlepšilo sa i logovanie a práca s výskytom neočakávaných chýb, čo bolo dosiahnuté zapojením cyklu odchytávania výnimiek, ktorý sprehľadnil proces a umožnil centralizovane spravovať problémové stavy. Stanovený vzor je ale potrebné dodržiavať, pretože bez toho tieto zmeny, prípadne dostupná metodika stráca zmysel. Popísané postupy a patterny zabezpečujú jednoduchšiu prácu, čo sa ocení najmä pri rozsiahlejších projektoch.

Po vykonaní testov a na základe meraní z následujúcej kapitoly bola znovu skounštruovaná kocka 7.1, ktorá ukazuje výrazné zlepšenie najmä pri parametroch rozšíriteľnosti a udržateľnosti. Žiadúci parameter bol i efektivita a použiteľnosť čo sa tiež podarilo vylepšiť. Parameter, ktorý sa veľmi nezlepšil bola spoľahlivosť resp. dostupnosť, to ale nebolo predmetom optimalizácií a je to spôsobené najmä občasnými výpadkami serverov.



Obr. 7.1: Prehľad kvalitatívnych parametrov po aplikovaní metodiky

7.5 Záver testovania

V závere možno konštatovať, že testy priniesli očakávané výsledky a teda správnosť hypotézy sa potvrdila. Podarilo sa formulovať metodiku, ktorej pochopením a správnym použitím je možné zlepšiť stav z hľadiska kvalitatívnych parametrov webových aplikácií.

Záver

Skutočnosť či vytvorené vzory a metodika sú prínosom i vo všeobecnosti je závislá najmä na schopnosti a motivácii vývojárov poskytnuté prostriedky použiť. Tak ako každá rada, môže byť akceptovaná, dobre aplikovaná a časom môže priniesť značný úžitok.

Musím ale konštatovať, že v prípade systému záverečných prác, bola aplikovaná metodika veľmi prínosná a pomocou nej sa stanovené ciele podarilo naplniť. Jej aplikáciou sa podarilo dosiahnuť, že súčasný stav je dobre škálovateľný a rozložený na nezávislé časti, ktoré sú dobre rozšíriteľné. Optimalizácia mala za následok aj zlepšený chod a odozvu systému z pohľadu užívateľov o čom svedčia i výsledky testov.

Osobný prínos

Praktické skúsenosti v akejkoľvek sfére v IT sú neodmysliteľnou súčasťou procesu vývoja každého vývojára. Teória je podstatná, no bez praxe je to vždy len stav hypotézy a subjektívneho názoru. V rámci tejto práce som si vyskúšal optimalizáciu už vydaného releasu aplikácie, ktorá sa vyvíjala za úplne iných podmienok a nástrojov aké sú dostupné dnes. Získal som skúsenosti ako správne manažovať svoj čas, keďže súčasne som pôsobil na ďalšom projekte postavenom na technológii IBM BPM. Pri práci v komunite vývojárov som mal možnosť nazrieť do častí, ktoré som doposiaľ neobjavil a teda moje skúsenosti v tejto oblasti rástli omnoho rýchlejšie.

Pohľad ďalej

V rámci ďalšieho rozvoja je cieľom vytvoriť vzorovú aplikáciu, ktorá začínajúcim vývojárom v IBM BPM poskytne komplexný prehľad možností tejto technológie a to formou sady tutoriálov a vzorových implementácií. Cieľom je centralizácia materiálov na výučbu či zdokonaľovanie sa v oblasti vývoja procesných aplikácií v IBM BPM a poskytnúť tak miesto odkiaľ je možné čerpať všetky potrebné informácie jednoducho a rýchlo.

Literatúra

[1] Ruiz, J.; Calero, C.; Piattini, M.: A Three Dimensional Web Quality Model1. [online], 2011, [cit. 8. 5. 2017]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-13911-6_15

[2] Polillo, R.: Quality in Web Engineering Workshop, A Methodological Approach and a Proposal. [online], 2011, [cit. 8. 5. 2017]. Dostupné z: https://pt.slideshare.net/rpolillo/quality-models-for-web-sites

[3] Hordějčuk, V.: Systémová architektura. [online], 2008, [cit. 8. 5. 2017]. Dostupné z: http://voho.eu/wiki/model-view-controller

[4] Ahukanna, D.; de Almeida, V. P. A.; Gucer, V.; aj.: IBM WebSphere Application Server V8 Concepts, Planning, and Design Guide. [online], 2013, [cit. 8. 5. 2017]. Dostupné z: http://www.redbooks.ibm.com/redbooks/pdfs/sg248135.pdf

[5] Schume, P.; Parrott, D.: BPM Layered Architecture Pattern. [online], 2015, [cit. 8. 5. 2017]. Dostupné z: https://developer.ibm.com/bpm/docs/functional-architecture/layered-architecture-pattern

[6] Murcko, T.: Business Dictionary. [online], 2016, [cit. 8. 5. 2017]. Dostupné z: http://www.businessdictionary.com/definition/website.html

[7] Systems, A.: Dynamické weby, stránky a webové formuláre. [online], 2016, [cit. 8. 5. 2017]. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html

[8] Lew, P.; Olsina, L.; Zhang, L.: Quality, Quality in Use, Actual Usability and User Experience as Key Drivers for Web Application Evaluation. [online], 2010, [cit. 8. 5. 2017]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-13911-6_15

[9] Björemo, M.; Trninić, P.: Evaluation of web application frameworks. [online], 2010, [cit. 8. 5. 2017]. Dostupné z: http://publications.lib.chalmers.se/records/fulltext/123847.pdf

[10] Madeyski, L.; Stochmialek, M.: Architectural design of modern web applications. [online], 2012, [cit. 8. 5. 2017]. Dostupné z: http://madeyski.e-informatyka.pl/download/23.pdf

[11] Lozano, M. D.; González, P.; Ramos, I.: A First Approach To Design Web Sites By Using Patterns. [online], 2002, [cit. 8. 5. 2017]. Dostupné z: https://www.researchgate.net/profile/Pascual_Gonzalez/publication/266871459_A_First_Approach_To_Design_Web_Sites_By_Using_Patterns/links/544e26b80cf29473161a1aed.pdf

[12] Gamma, E.; Helm, R.; Johnson, R.; aj.: Elements of Reusable Object-Oriented Software. [online], 2007, [cit. 8. 5. 2017]. Dostupné z: http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf

[13] Kotúč, J.: Zátěžové testy webových aplikací v kontextu průběžné integrace. [online], 2010, [cit. 8. 5. 2017]. Dostupné z: https://is.muni.cz/th/331131/fi_m/dp_jan_kotuc.pdf

[14] Menascé, D. A.: Load Testing of Web Sites. [online], 2002, [cit. 8. 5. 2017]. Dostupné z: http://cs.gmu.edu/~menasce/papers/IEEE-IC-LoadTesting-July-2002.pdf

[15] Ticknor, M.; Corcoran, A.; Csepregi-Horvath, B.; aj.: IBM WebSphere Application Server V8 Concepts, Planning, and Design Guide. [online], 2011, [cit. 8. 5. 2017]. Dostupné z: http://www.redbooks.ibm.com/redbooks/pdfs/sg247957.pdf

[16] Silver, D. B.; Fischer, L.: BPMN 2.0 Handbook Second Edition. [online], 2012, [cit. 8. 5. 2017]. Dostupné z: http://www.conradbock.org/white-bpmn2-process-bookmark-web.pdf

[17] Kolban, N.: Kolban's book on IBM Business Process Management. [online], 2014, [cit. 8. 5. 2017]. Dostupné z: http://neilkolban.com/ibm/wp-content/uploads/2014/12/Kolbans-IBPM-Book-2014-12.pdf

[18] Collins, M.; Duan, Z. H.; Fried, A.; aj.: IBM Business Process Manager V8.5 Performance Tuning and Best Practices. [online], 2015, [cit. 8. 5. 2017]. Dostupné z: http://www.redbooks.ibm.com/redbooks/pdfs/sg248216.pdf

Dodatok **A**

Zoznam použitých skratiek

AJAX Asynchronous Javascript and XML

API Application Programming Interface

BPM Business Process Manager

BPMN Business Process Manager Notation

CV Coach View

CZM Centrum znalostního managementu

HS Human Service

IBM International Business Machine

IBPM IBM BPM

MVC Model View Controller

MVCS Model View Controller Service

OS Operačný systém

OOP Obiektovo-orientované programovanie

TK Toolkit

UCA Undercover Agent

Dodatok **B**

Slovník pojmov

Backend časť webovej aplikácie, ktorá obplyvňuje funkcionalitu a nie je viditeľná bežným užívateľom

Bug chyba

Cache pomocná pamäť, ktorá si ukladá dočasne dáta, aby skrátilo čakaciu dobu, často oproti serveru

Flow Tok, Napr. procesný flow, procesný tok, flow dát

Frontend časť webovej aplikácie vidieľná bežným návštevníkom

Learning curve grafické vyobrazenie plynulosti postupu pri učení sa

Logy zátnamy zo servera o činnosti aplikácie

Release nasadenie aplikácie do produkčného prostredia, napr. 1. produkčný release

String datový typ, textový reťazec

Timeout uplynutie, vypršanie určitého časového limitu

Query otázka, spravidla príkaz oproti databáze

Dodatok C

XLS šablóna vstupu transformačnej služby

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template name="format-date-time">
            <xsl:param name="date" />
            <xsl:value-of select="substring($date, 1, 4)" />
            <xsl:text>/</xsl:text>
            <xsl:value-of select="substring($date, 6, 2)" />
            <xsl:text>/</xsl:text>
            <xsl:value-of select="substring($date, 9, 2)" />
            <xsl:text> </xsl:text>
            <xsl:text>00</xsl:text>
            <xsl:text>:</xsl:text>
            <xsl:text>00</xsl:text>
            <xsl:text>:</xsl:text>
            <xsl:text>00</xsl:text>
            <xsl:text>.0 UTC</xsl:text>
        </xsl:template>
      <xsl:template match="entry">
        <variable type="Semester">
            <code type="String"><xsl:value-of select="content/code" /></code>
            <name_cs type="String"><xsl:value-of select="content/name[@lang='cs']"
                 /></name_cs>
            <name_en type="String"><xsl:value-of select="content/name[@lang='en']"
                 /></name_en>
            <xsl:choose>
                <xsl:when test="content/startDate!=''">
                    <startDate type="Date">
                        <xsl:call-template name="format-date-time">
                            <xsl:with-param name="date" select="content/startDate"
                                 />
                        </xsl:call-template>
                    </startDate>
                </xsl:when>
                <xsl:otherwise></xsl:otherwise>
            </xsl:choose>
            <xsl:choose>
                <xsl:when test="content/endDate!=''">
                    <endDate type="Date">
                        <xsl:call-template name="format-date-time">
                            <xsl:with-param name="date" select="content/endDate" /
                                 >
                        </xsl:call-template>
                    </endDate>
                </xsl:when>
                <xsl:otherwise></xsl:otherwise>
            </xsl:choose>
            </variable>
    </xsl:template>
</xsl:stylesheet>
```

Dodatok D

Diagram CVs troch optimalizovaných modulov

Obr. D.1: Náhľad obsahujúci všetky komponenty príslušných obrazoviek v rámci všetkých optimalizovaných aplikácií

Dodatok **E**

Obsah priloženého CD

- `attachments` ..... zdrojové súbory z príloh
  - `cv_diagram.jpg` ..... Diagram CV aplikácií
  - `transform_template.xsl` ..... Transformačná XSL šablona
- `images` ..... adresár s obrázkami
- `mybibliographyfile.bib` ..... súbor citovaných zdrojov
- `prokiiva_DP_2017.pdf` ..... text práce ve formáte PDF
- `prokiiva_DP_2017.tex` ..... zdrojová forma práce vo formáte LaTeX
- `readme.txt` ..... stručný popis obsahu CD