

UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED

**MANAŽÉRSTVO KVALITY S VYUŽITÍM PODNIKOVÝCH
INFORMAČNÝCH SYSTÉMOV**
DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	PaedDr. Viera Michaličková, PhD.
Konzultant:	Ing. Anton Smutný

Nitra 2016

Bc. Tomáš Tóth



Univerzita Konštantína Filozofa v Nitre
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Tóth
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9 aplikovaná informatika
Typ záverečnej práce: Diplomová práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Manažérstvo kvality s využitím podnikových informačných systémov

Anotácia: Manažérstvo kvality je dôležitou súčasťou riadenia podniku. Jeho cieľom je optimalizácia pracovných postupov alebo výrobných procesov so zohľadnením úspory zdrojov, očakávanej kvality produktu alebo poskytovanej služby a predpokladaného ďalšieho rastu a konkurencieschopnosti firmy. Podnikový informačný systém je významným nástrojom zvyšovania efektívnosti najrôznejších činností v podniku, vrátane zabezpečovania kvality.

Ciele diplomovej práce:

analyzovať aktuálny stav v oblasti manažmentu kvality v podnikoch v súvislosti s využívaním podnikových informačných systémov,
navrhnuť softvérové riešenie (so zameraním sa na jeho backendovú časť) pre oddelenie kvality v konkrétnej firme,
riešenie implementovať s využitím technológie Java EE,
na základe vykonaných testov zhodnotiť nasadenie vyvinutého systému v praxi.

Požiadavky na vedomosti a zručnosti študenta:

analytické schopnosti,
programátorské zručnosti,
skúsenosti s technológiou Java a webovými aplikáciami.

Školiteľ: PaedDr. Viera Michaličková, PhD.
Oponent: Mgr. Martin Vozár, PhD.
Konzultant: Ing. Anton Smutný
Katedra: KI - Katedra informatiky

Dátum zadania: 02.10.2014

Dátum schválenia: 27.10.2014


prof. Ing. Milan Turčáni, CSc.
schválil/a

POĎAKOVANIE

Rád by som sa poďakoval mojej školiteľke, PaedDr. Viere Michaličkovej, PhD., za hodnotné rady, odporúčania a odborné vedenie pri vypracovávaní diplomovej práce. Tiež by som sa chcel poďakovať aj Ing. Antonovi Smutnému a vybranej spoločnosti za poskytnutú príležitosť vyvíjať softvér v praxi a možnosť získať tak cenné skúsenosti. Ďakujem aj Ing. Marekovi Dianovi a Ing. Jozefovi Machovi za profesionálne rady pre vývoj kvalitného softvéru.

ABSTRAKT

TÓTH, Tomáš: Manažérstvo kvality s využitím podnikových informačných systémov. [Diplomová práca]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied. Školiteľ: PaedDr. Viera Michaličková, PhD. Konzultant: Ing. Anton Smutný. Stupeň odbornej kvalifikácie: Magister odboru Aplikovaná informatika. Nitra: FPV, 2016. 87 s.

Vzhľadom na zvyšujúci sa dôraz kladený na kvalitu výrobkov a služieb sa manažérstvo kvality a jej zabezpečovanie stáva nevyhnutnou súčasťou dnešných podnikov. Taktiež efektívnemu vykonávaniu procesov sa prisudzuje stále väčšia dôležitosť v dnešnom rýchlo sa rozvíjajúcom a konkurenčnom prostredí podnikov. V diplomovej práci sa zaoberáme zefektívnením a zautomatizovaním procesov v manažérstve kvality v podnikoch vybranej spoločnosti, pričom využívame informačné technológie a podnikové informačné systémy. V teoretickej časti popisujeme manažérstvo kvality a jeho význam. Tiež sa zaoberáme informačnými systémami a ich vývojom. V diplomovej práci uvádzame prípadovú štúdiu manažérstva kvality v konkrétnom podniku. V praktickej časti sa zameriavame práve na vývoj podnikového informačného systému pre podporu procesov manažérstva kvality, opisujeme postup vývoja na platforme Java EE, pričom sa zameriavame najmä na backendovú časť. Venujeme sa aj výkonnostným testom a spôsobu pripojenia na SAP ERP. Následne hodnotíme vyvinutý podnikový informačný systém a aj jeho využitie v praxi.

Kľúčové slová: Podnikové informačné systémy. Manažérstvo kvality. Java EE.

ABSTRACT

TÓTH, Tomáš: Quality management using enterprise information systems. [Diploma Thesis]. Constantine the Philosopher University in Nitra. Faculty of Natural Sciences. Supervisor: PaedDr. Viera Michaličková, PhD. Consultant: Ing. Anton Smutný. Degree of Qualification: Magister of Applied Informatics. Nitra: FNS, 2016. 87 p.

With the increasing emphasis on quality of products and services, quality management and its assurance becomes an essential part of today's enterprises. Importance is also attributed to effective implementation of processes in today's fast evolving and competitive environment of businesses. The diploma thesis deals with processes in quality management in enterprises of specific company. We want to increase the efficiency of processes implementation or make them fully automated, taking advantage of information technology and enterprise information systems. The theoretical part describes the quality management and its importance. We also deal with information systems and their development. In the diploma thesis we present a case study of quality management in the company. The practical part focuses on the development of enterprise information system to support the quality management processes. We describe the process of development on the Java EE platform, focusing mainly on the backend part. We also deal with the performance tests and how to connect to SAP ERP. Subsequently we evaluate the developed enterprise information system and also its use in practice.

Keywords: Enterprise information systems. Quality management. Java EE.

OBSAH

Úvod	9
1 Analýza súčasného stavu	10
1.1 Manažérstvo kvality v podnikoch	10
1.1.1 Definícia pojmu kvalita	11
1.1.2 Vymedzenie pojmov v oblasti manažérstva kvality	12
1.1.3 Kontrola kvality	14
1.1.4 Normy v oblasti manažérstva kvality	16
1.1.5 Certifikácie a posudzovanie zhody	17
1.2 Podnikové informačné systémy	17
1.2.1 Vymedzenie základných pojmov v informačných systémoch.....	18
1.2.2 Informačný systém	19
1.2.3 Systém pre plánovanie zdrojov (ERP)	21
1.2.4 Výskum v oblasti implementácie a využívania ERP systémov	24
1.2.5 Projektovanie podnikového informačného systému.....	27
1.2.6 Životný cyklus informačného systému	29
1.2.7 Modely životného cyklu informačných systémov.....	31
1.2.8 Metodiky vývoja softvéru	34
2 Ciele diplomovej práce	39
3 Vývoj podnikového informačného systému.....	40
3.1 Manažérstvo kvality vo vybranom podniku (prípadová štúdia).....	40
3.1.1 Vytvorenie doménového slovníka	41
3.1.2 SAP ERP	42
3.1.3 Proces výroby a kontroly kvality	42
3.2 Špecifikácia požiadaviek.....	43
3.3 Návrh PIS QAM.....	45
3.4 Architektúra vyvíjaného PIS QAM.....	46
3.5 Návrh a vytvorenie databázy.....	48

3.6	Návrh tried	49
3.7	Java Persistence API (JPA)	49
3.8	JPA entity	50
3.9	Návrhový vzor Data Access Object (DAO)	52
3.10	Práca s entitami	52
3.10.1	EntityManager	53
3.10.2	JPQL dotazy	53
3.11	Komunikácia backendu s frontendom	55
3.11.1	Rozhranie biznis vrstvy	55
3.11.2	Návrhový vzor Data Transfer Object (DTO)	56
3.12	Grafické používateľské rozhranie	56
3.12.1	JSF, PrimeFaces, Ajax	56
3.12.2	Vývoj GUI pre PIS QAM	57
3.13	Prehľad zákaziek	57
3.13.1	Prehľad zákaziek podľa stavu zákazky	58
3.13.2	Manuálne vytvorenie zákazky	59
3.13.3	Detail zákazky a jeho editácia	60
3.13.4	Zjednotenie formátu dátumov a času	61
3.14	Zaznamenávanie vykonaných akcií	62
3.15	Jazykové mutácie	63
3.16	Protokoly	64
3.16.1	Prehľad požadovaných protokolov	64
3.16.2	Implementácia funkcionality práce s protokolmi	65
3.16.3	Kontrola procesov a informovanie pracovníka	67
3.17	Test výkonnosti a optimalizácia	67
3.17.1	Interceptor na meranie času	67
3.17.2	Generovanie entít v databáze	69
3.17.3	Sledovanie zaťaženia databázy	70

3.17.4	Ladenie výkonu	71
3.18	Prepojenie so SAP ERP	72
3.18.1	SAP Java Connector (SAP JCo)	73
3.18.2	Implementácia pripojenia na SAP ERP v Jave	73
3.18.3	Funkčné moduly	74
3.18.4	Konvertovanie z JCoTable	75
3.18.5	Plánovanie úloh na pozadí	76
3.19	Dokumentovanie systému	77
4	Výsledky riešenia a ich zhodnotenie.....	78
	Záver	82
	Zoznam bibliografických odkazov	83
	Zoznam príloh	87

ÚVOD

V dnešnom svete môžeme vidieť, ako rýchly vývoj vplýva na každú oblasť nášho života. Menia sa technológie, ale aj procesy vykonávania činností v rôznych sférach. Podniky sa v súčasnosti nachádzajú v globálnom prostredí, ktoré taktiež podlieha rýchlo sa vyskytujúcim zmenám. Aby boli podniky schopné uspieť v konkurenčnom boji, musia byť schopné reagovať na tieto zmeny. Zo strany zákazníkov sa zároveň vyžaduje kvalita služieb a výrobkov, čo môže byť tiež rozhodujúcim faktorom pri úspechu. Podniky musia klásť stále väčší dôraz na kvalitu poskytovaných služieb a vyrábaných výrobkov, aby splnili zákazníkove požiadavky a uspeli. Manažérstvo kvality sa tak stáva nevyhnutnou súčasťou dnešných podnikov.

Nemenej dôležité je efektívne vykonávanie pracovných činností. Aj v tomto smere môžu byť nápomocné informačné technológie a informačné systémy, ktoré dokážu pomôcť používateľovi nielen pri práci s veľkým množstvom dát. Takáto situácia s požiadavkou na vytvorenie informačného systému sa vyskytla aj v nami vybranej spoločnosti a jej podnikoch. Zámerom tejto diplomovej práce je vytvorenie podnikového informačného systému na podporu pracovných procesov v podniku na oddelení zabezpečovania kvality.

V teoretickej časti diplomovej práce popisujeme manažérstvo kvality a jeho význam v podnikoch. V tejto časti sa zaoberáme aj informačnými systémami a ich vývojom. V praktickej časti sa zameriavame práve na vývoj podnikového informačného systému pre vybranú spoločnosť a jej podniky. Uvádžame prípadovú štúdiu manažérstva kvality v podniku spoločnosti. Na základe prebiehajúcich procesov v podniku navrhujeme a následne aj vytvárame riešenie v podobe špecializovaného podnikového informačného systému určeného na podporu týchto procesov. Pri vývoji sme použili platformu Java Enterprise Edition a iné technológie. Pri popise vývoja sa zameriavame najmä na backendovú časť informačného systému. Popisujeme architektúru vyvíjaného informačného systému, prácu s dátami, funkcionality systému, tvorbu dokumentácie s využitím reverzného inžinierstva a priebeh testovania výkonnosti informačného systému s jeho následnou optimalizáciou. Venujeme sa aj implementácii pripojenia vyvíjaného informačného systému na celopodnikový systém SAP ERP za účelom automatizovaného získavania dát.

Následne hodnotíme celý vyvinutý informačný systém, jeho funkcie a možnosti jeho využitia v praxi. Taktiež hodnotíme aj postupy, ktoré sme si zvolili pri jeho vývoji.

1 ANALÝZA SÚČASNÉHO STAVU

Zabezpečenie konkurencieschopnosti a trvalo udržateľného rozvoja podnikov, či už na národnom alebo medzinárodnom trhu, si v súčasných podmienkach vyžaduje riešenie nových otázok, nové prístupy, nové nástroje, nové myslenie a nové informácie. Úspešnosť fungovania každého z nich vo veľkej miere závisí od rýchlosti a schopnosti prispôbiť sa požiadavkám a prániam jednotlivých zákazníkov. V silnom konkurenčnom prostredí je pre zabezpečenie dlhodobej prosperity potrebné v procese riadenia sledovať značné množstvo aspektov, čo sa nezaobíde bez aplikácie informačných systémov. Taktiež požiadavky na vlastnosti výrobkov so zaskomponovaním špecifických požiadaviek zákazníkov získavajú stále väčší význam. Kvalita je v súčasnom konkurenčnom boji na trhu nevyhnutnosťou pre získanie, resp. udržanie si podielu na trhu. Aby boli podniky konkurencieschopné a udržali si hospodársku výkonnosť, začali zavádzať efektívne systémy manažérstva kvality (Hrubec, 2001; Košovská, Váryová, Ferenczi Vaňová, 2013).

Diplomová práca je zameraná na vývoj podnikového informačného systému pre manažérstvo kvality, preto v nasledujúcich podkapitolách popisujeme manažérstvo kvality v podnikoch a zabezpečovanie kvality pri výrobnom procese, analyzujeme problematiku informačných systémov v podnikoch, pričom sa zameriavame aj na ich vývoj.

1.1 MANAŽÉRSTVO KVALITY V PODNIKOKCH

Činnosť priemyselného podniku je zložitým transformačným procesom, pri ktorom vstup do podniku (materiál, suroviny) sa mení na požadovaný výstup (hotové výrobky). Na splnenie plánovaných cieľov musí podnik vykonávať súbor rôznych čiastkových procesov (činností), z ktorých každý vyžaduje riadenie. Pokiaľ sa pri celkovom riadení pôsobí na správanie podniku ako celku, riadenie jednotlivých subsystémov (funkčných oblastí) vyžaduje riadiace postupy, ktoré sú vlastné len konkrétnemu subsystému. K takým subsystémom riadenia podniku patrí aj subsystém riadenia kvality – *manažérstvo kvality* (Šatanová, Gejdoš, 2007).

Ako uvádza Harausová (2012), kvalitný produkt môže vzniknúť len v dobre riadenom a organizovanom podniku. Preto by malo byť v záujme každého podniku zaviesť, udržiavať a zdokonaľovať systém manažérstva kvality. Manažérstvo kvality je

v súčasnosti nevyhnutnosť, ktorá môže zabezpečiť dlhodobú konkurenčnú schopnosť podniku.

1.1.1 Definícia pojmu kvalita

V našej slovenskej odbornej literatúre, ale i v praxi sa bežne okrem pojmu *kvalita* používa tiež pojem *akosť*. Z jazykového hľadiska ide o synonymá a v spisovnej slovenčine je povolené používanie oboch pojmov. V slovenských technických normách (STN) sa používa pojem *kvalita výrobku*, preto aj my ho budeme ďalej používať (Šatanová, Gejdoš, 2007).

Paulová (2013) uvádza, že definícia pojmu kvalita sa vyvíjala historicky ako určitá predstava vnímania a očakávania od konečných výstupov – produktov, ktoré možno identifikovať ako určitý konkrétny výrobok (hmotný výstup) alebo poskytnutie služby.

Kvalita sa vo výrobkovo orientovaných definíciách zjednodušene posudzuje, ako zhoda vlastností výrobku s predpísanou technickou dokumentáciou, t.j. ako bezchybnosť. Zabezpečovala sa napr. následnou kontrolou podľa stanovených technických pravidiel a predpisov. Ak bol podľa nich výrobok označený za bezchybný, bol prehlásený za kvalitný (Šatanová, Gejdoš, 2007).

Harausová (2012) charakterizuje kvalitu ako súhrn vlastností, znakov a charakteristík výrobku alebo služby, ktorý im dáva schopnosť uspokojovať vopred stanovené alebo predpokladané potreby. Kvalita je spokojnosť všetkých: zákazníkov, vlastníkov a zamestnancov.

Slovenská technická norma STN 01 0101 definuje kvalitu nasledovne: „súhrn vlastností vyjadrujúcich spôsobilosť plniť funkcie, pre ktoré je určený, súčasne sa berú do úvahy i ekonomické ukazovatele výrobku, jeho vybavenie príslušenstvom, náhradnými dielmi a podobne, ako i predpoklady, ktoré výrobca vytvára pre poskytovanie služieb spojených s používaním výrobku“ (Šatanová, Gejdoš, 2007).

Keďže cieľom tejto záverečnej práce je aj vytvoriť podnikový informačný systém pre podniky, ktoré pôsobia nielen na Slovensku, ale aj v Nemecku, uvedieme definíciu kvality aj podľa nemeckej normy DIN 55 350. Nemecká norma DIN 55 350 definuje kvalitu ako súhrn vlastností a znakov výrobku alebo činnosti vzťahujúcej sa prostredníctvom jej spôsobilosti k uspokojovaniu daných potrieb (Šatanová, Gejdoš, 2007).

Vývoj manažerstva kvality vo svete významne ovplyvnilo Japonsko, uvádzame preto aj definíciu podľa japonských priemyselných noriem, kde je kvalita definovaná ako „súhrn príslušných znakov a charakteristík, ktoré sú predmetom ocenenia, ktorého cieľom je stanoviť, či výrobok alebo služba vyhovujú účelu svojho použitia alebo nie“ (Mateides, Styk, Kučera, 1998).

1.1.2 Vymedzenie pojmov v oblasti manažerstva kvality

V nasledujúcich riadkoch popíšeme termíny a definície podľa medzinárodnej normy ISO 9000:2015 Quality management systems — Fundamentals and vocabulary (2015). V niektorých publikáciách vydaných pred rokom 2015 sa v tomto ohľade môže spomínať norma ISO 9000:2005 alebo ISO 9000:2000. Norma ISO 9000:2015 je najnovším vydaním a ruší a nahrádza predchádzajúce vydania (ISO 9000:2005, ISO 9000:2000).

Kvalita je miera s akou súbor vlastných charakteristík spĺňa požiadavky.

Systém je súbor vzájomne prepojených alebo vzájomne pôsobiacich prvkov.

Systém manažerstva je systém na určenie politiky, cieľov a postupov na dosiahnutie týchto cieľov. Politiku kvality tvoria celkové zámery a pôsobenia organizácie v oblasti kvality.

Systém manažerstva kvality je systém manažerstva na riadenie a usmerňovanie organizácie vzhľadom na kvalitu.

Manažerstvo kvality charakterizujeme ako koordinované činnosti zamerané na riadenie a usmerňovanie organizácie vzhľadom na kvalitu. Manažerstvo kvality zahŕňa pododvetvia ako je politika kvality, ciele kvality, plánovanie kvality, riadenie kvality, zabezpečovanie kvality a zlepšovanie kvality. V niektorých zdrojoch sa môžeme stretnúť aj s termínom „manažment kvality“, ale keďže v slovenskej technickej norme STN EN ISO 9000 Systémy manažerstva kvality. Základy a slovník (ISO 9000: 2015) (ÚNMS SR, 2016) je tento termín definovaný ako „manažerstvo kvality“, rozhodli sme sa ho aj my v tomto znení používať.

Svojimi výsledkami v oblasti manažerstva kvality si Japonsko vydobylo vysoký medzinárodný štandard. Manažerstvo kvality v Japonsku má charakter komplexne organizovaného systému, ktorý vychádza z predpokladu, že sa na ňom organizačne podieľajú všetky útvary a kategórie zamestnancov od vrcholového vedenia až po robotníkov, pričom v japonských priemyselných spoločnostiach vedenie nedeleguje svoju právomoc v rozhodovaní o kvalite výrobkov na odborné útvary riadenia kvality

v takej miere, ako je to v iných krajinách, i u nás. Na označenie takéhoto systému manažérstva kvality sa používa slovné spojenie *komplexné (totálne) manažérstvo kvality (TQM - Total Quality Management)* (Šatanová, Gejdoš, 2007).

Ciele kvality charakterizujeme ako niečo, o čo sa usilujeme v súvislosti s kvalitou. Zvyčajne sú založené na politike kvality organizácie.

Plánovanie kvality je časť manažérstva kvality, ktorá je zameraná na stanovovanie cieľov kvality a špecifikuje procesy, ktoré sú nevyhnutné na splnenie týchto cieľov.

Riadenie kvality je časť manažérstva kvality, ktorá je zameraná na splňanie požiadaviek kvality. Manažérstvo kvality a riadenie kvality môžu v slovenčine vyznieť ako synonymá, preto je potrebné byť pri prekladoch pozorný a rozlišovať originálne anglické termíny „quality management“ (manažérstvo kvality) a „quality control“ (riadenie kvality).

Zlepšovanie kvality je časť manažérstva kvality zameraná na zvyšovanie schopnosti splnenia požiadaviek kvality.

Zabezpečovanie kvality je časť manažérstva kvality zameraná na poskytovanie dôvery, že požiadavky na kvalitu budú splnené.

Mateides, Styk a Kučera (1998) uvádzajú, že zabezpečovanie kvality znamená zaručenie kvality výrobku, ktorý si zákazník môže s dôverou kúpiť a užívať ho ku svojej spokojnosti. Tiež definujú zabezpečovanie kvality ako súbor plánovaných a systematických aktivít zameraných na získanie potrebnej dôvery, že výrobok je v zhode s požiadavkami zákazníka. Táto definícia sa dotýka najmä činností v samotnom výrobnom procese a pri kontrole.

Zabezpečovanie kvality už dnes nespočíva len v prevencii, aby nekvalitný výrobok nevyšiel z podniku, ale v tom, aby nekvalitný výrobok nemohol byť vôbec vyrobený. Požiadavky zákazníkov zároveň zdôrazňujú potrebu stability kvality dodávaných výrobkov. Odberatelia vyžadujú od svojich dodávateľov, aby preukázali, že poskytujú záruky trvalého dosahovania proklamovanej úrovne kvality – musia dokázať existenciu spoľahlivého a fungujúceho systému kvality (Šatanová, Gejdoš, 2007).

Systémy manažérstva kvality výraznou mierou vplývajú na kvalitu vyrobeného výrobku, a tak môžu pomôcť organizáciám zvýšiť spokojnosť zákazníka. Pretože potreby a očakávania zákazníka sa menia a spôsobujú konkurenčný tlak, organizácie sú nútené trvalo zlepšovať svoje produkty a procesy (Hrubec, 2001).

1.1.3 Kontrola kvality

Medzi najdôležitejšie činnosti v manažérstve kvality patrí *kontrola kvality*. Kontrola kvality je činnosť, ktorá zabezpečuje úroveň a mieru kvalitných vlastností kontrolovaného predmetu. Zistené výsledky porovnáva s dopredu stanovenými požiadavkami a podľa výsledku porovnávania rozhoduje, či kontrolovaný predmet je, alebo nie je vhodný pre daný účel. Kontrole podliehajú pracovné predmety vo všetkých štádiách rozpracovanosti (materiál, polotovary a hotové výrobky), kontrolné prostriedky (meradlá, meracie prístroje, výrobné zariadenia ako sú stroje, nástroje a aparatury). Kontrole podliehajú všetky produkty – hmotné aj nehmotné (Šatanová, Gejdoš, 2007).

Vo výrobnom procese nedokážeme nikdy vyrobiť dva rovnaké výrobky. Rozdiely môžu byť malé alebo veľké, ale stále existujú. Rozdiely, ktoré zistíme pri sledovaní určitého znaku (rozmer, drsnosť, tvrdosť,...) sa nazývajú rozptyl, ktorý sa stáva predmetom sledovania procesu. Pri získavaní presného prehľadu o rozptyle by bolo potrebné zmerať každý výrobok z daného procesu. Tento postup je však časovo náročný a príliš drahý. Preto sa robí náhodná kontrola vybraných vzoriek a získané výsledky umožňujú na základe štatistických postupov urobiť predpovede pre celý súbor. Presnosť predpovede závisí od množstva odobratých vzoriek a použitej štatistickej metódy. Kvalita nameraných hodnôt a z nich vypočítané charakteristiky majú rozhodujúci vplyv na vykonanie nápravných činností v procese (Hrubec, 2001).

Existuje niekoľko typov kontrol, ktorých podrobný popis by ale presahoval možnosti tejto záverečnej práce, preto si ich vymenujeme aspoň pár.

Podľa toho, či človek posudzuje kvalitu sám, alebo pomocou meradiel a meracích prístrojov, rozlišujeme kontrolu:

- subjektívnu – spočíva v zmyslovom hodnotení a týka sa výrobkov, pre ktoré sú zmyslové vlastnosti dôležitou súčasťou kvality (potravínarske a spotrebné výrobky, estetické vlastnosti všetkých ostatných výrobkov),
- objektívnu – spočíva v posudzovaní kvantitatívnych vlastností výrobkov; podľa presnosti tohto posúdenia delíme objektívnu kontrolu na kontrolu:
 - meraním – pri kontrole tohto typu dostaneme číselnú hodnotu kvalitatívnej vlastnosti (znaku),
 - porovnávaním – výrobky sa delia na vyhovujúce a nevyhovujúce,

- triedením do skupín – triedenie sa môže uskutočňovať porovnávaním alebo meraním; na meracom prístroji je nastavená jedna alebo viac veličín a podľa ich dosiahnutia alebo prekročenia sa výrobky roztriedia.

Podľa konštrukcie kontrolných zariadení rozlišujeme v podstate tieto základné smery mechanizácie a automatizácie kontrolných operácií:

- kontrolné prvky na meranie jedného rozmeru – využívajú sa individuálne meradlá (posuvné meradlá, kalibre),
- niekoľko rozmerové meradlá – umožňuje súbežnú kontrolu niekoľkých rozmerov, čo podstatne skracuje čas kontroly; používajú sa rôzne mechanické, elektronické a pneumatické meracie prístroje,
- poloautomatické meracie zariadenia – sú charakterizované tým, že okrem mechanického merania je tu navyše mechanizácia vkladania výrobkov do meracieho prístroja, prípadne i triedenie výrobkov je mechanizované,
- automatizované meracie a triediace stroje – predstavujú najvyšší stupeň nahradenia ľudskej práce pri kontrole; používajú sa hlavne v hromadnej výrobe jednoduchých výrobkov.

Podľa rozsahu kontroly rozlišujeme kontrolu:

- stopercentnú (úplnú) – kontrole podliehajú všetky výrobky; uplatňuje sa predovšetkým pri kontrole malých súborov dôležitých a drahých výrobkov a v bežnej hromadnej výrobe v prípadoch, kde môžeme použiť automatické meracie a triediace stroje,
- výberovú – nekontroluje všetky výrobky, iba ich časť; člení sa na kontrolu:
 - náhodnú – nemá predpísaný žiadny záväzný postup pre určenie rozsahu výberu ani pre rozhodnutie o tom, kedy výsledok kontroly možno považovať za uspokojivý; preto sa táto kontrola využíva len ako doplnok ostatných foriem kontroly a ako predbežná orientačná pomôcka,
 - pevného percenta výrobkov – je najpoužívanejším druhom čiastočnej kontroly; pri rôznom objeme kontrolovaných súborov zostáva kontrolovaný podiel alebo percento nemenné, čo spôsobuje, že presnosť kolíše podľa veľkosti kontrolovaného súboru alebo množstva,
 - štatistickú výberovú kontrolu – je založená na matematickej štatistike a počte pravdepodobnosti; v porovnaní s ostatnými formami kontroly je najpresnejšia a najekonomickejšia; aj pri vysokých požiadavkách na

kvalitu zabezpečí správnosť výsledkov s veľkou dopredu zvolenou pravdepodobnosťou (Šatanová, Gejdoš, 2007).

1.1.4 Normy v oblasti manažérstva kvality

Technická normalizácia je tvorivá činnosť, ktorou sa pre opakujúce sa technické úlohy zabezpečuje, určuje a uplatňuje najvýhodnejšie riešenie hlavne z hľadiska hospodárnosti, kvality a bezpečnosti. Každý normalizovaný výrobok musí mať určitú kvalitu, čo v praxi znamená, že musí vyhovovať určitým požiadavkám z hľadiska svojej funkcie (Mateides, Styk, Kučera, 1998).

Začiatkom osemdesiatych rokov sa rozhodla Medzinárodná organizácia pre normalizáciu ISO zaoberať problematikou normalizácie v oblasti kvality. V roku 1979 založila technickú komisiu ISO/TC 176 Manažérstvo kvality a zabezpečovanie kvality, ktorá v priebehu svojej činnosti vydala súbor noriem radu ISO 9000. Tento súbor noriem podpísalo 72 priemyselne najvyspelejších krajín. Normy boli akýmsi návodom na vytvorenie a určenie pravidiel riadenia činností, ktoré ovplyvňujú kvalitu výstupného produktu. Normy boli nezáväzné, ak ich však organizácia prijala za svoj základ, bolo povinnosťou ich v plnej miere naplňať (Hrubec, 2001; Paulová, 2013).

V názvoch noriem sa môžeme stretnúť okrem označenia ISO aj s označeniami ako je EN a STN. EN je európska norma, ktorá predstavuje ISO normu prispôbenú na európske podmienky. STN je slovenská technická norma založená na EN.

Základ normatívneho zabezpečenia systému manažérstva kvality tvoria normy ISO radu 9000, ktorým predchádzal vznik noriem AQAP (normy určené pre armádu). Normy boli postupom času pretransformované pre špecifické oblasti priemyslu, ako napr. automobilový, farmaceutický, potravinársky a pod. Normy ISO radu 9000 vytvárajú súvisiacu skupinu noriem systému manažérstva kvality a uľahčujú vzájomné pochopenie v národnom a medzinárodnom obchode. Patria sem:

- ISO 9000: Systémy manažérstva kvality. Základy a slovník. – opisuje základy systémov manažérstva kvality a špecifikuje terminológiu pre systémy manažérstva kvality,
- ISO 9001: Systém manažérstva kvality. Požiadavky. – špecifikuje požiadavky na systém manažérstva kvality, ktoré sú všeobecné a použiteľné v organizáciách akéhokoľvek priemyselného alebo ekonomického odvetvia bez ohľadu na ponúkanú kategóriu produktov; používa sa tam, kde organizácia potrebuje preukázať svoju schopnosť poskytovať produkty, ktoré spĺňajú požiadavky

zákazníka a použiteľných predpisov; zameriava sa na zdôraznenie spokojnosti zákazníka,

- ISO 9004: Systém manažérstva kvality. Manažérstvo trvalého úspechu organizácie. Prístup na základe manažérstva kvality. – poskytuje návod, ktorý berie do úvahy efektívnosť aj účinnosť systému manažérstva kvality; cieľom tejto normy je zlepšovanie výkonnosti organizácie, spokojnosť zákazníkov a ďalších zainteresovaných strán (ISO, 2005; Hrubec, 2001; Paulová, 2013).

1.1.5 Certifikácie a posudzovanie zhody

Hrubec (2001) uvádza, že *certifikácia* je činnosť certifikujúceho orgánu, pri ktorej sa zisťuje a vydaním certifikátu osvedčuje, že výrobok, systém kvality alebo odbornosť pracovníka je v zhode s požiadavkami predpisov alebo s dohodnutými resp. deklarovanými znakmi alebo vlastnosťami.

Iná definícia hovorí, že certifikácia výrobkov je činnosť tretej osoby preukazujúca dosiahnutie primeranej dôvery, že náležite identifikovaný výrobok, postup alebo služba je v zhode s predpísanou normou alebo iným normatívnym dokumentom (Mateides, Styk, Kučera, 1998).

Úrad pre normalizáciu, metrológiu a skúšobníctvo Slovenskej republiky definuje posudzovanie zhody ako zisťovanie, či skutočné vlastnosti určeného výrobku zodpovedajú ustanoveným technickým požiadavkám na určený výrobok. Zhoda sa považuje za preukázanú, ak určený výrobok spĺňa všetky technické požiadavky, ktoré sa naň vzťahujú.

Medzinárodná organizácia pre normalizáciu ISO uvádza, že certifikácia môže byť užitočným nástrojom pre zvýšenie dôveryhodnosti preukázaním, že produkt alebo služba spĺňa očakávania zákazníkov. V niektorých odvetviach je certifikácia právnou alebo zmluvnou povinnosťou.

1.2 PODNIKOVÉ INFORMAČNÉ SYSTÉMY

Vyspelosť systému riadenia podniku je daná tým, ako tento podnik dokáže zavádzať a využívať informačné technológie do svojho systému riadenia. Vyspelý podnik, ktorý trvalo používa informačné technológie, ich môže oveľa rýchlejšie využiť pre dosiahnutie svojich cieľov a tým pre zvýšenie konkurencieschopnosti (Kučera, Látečková, 2004). Harausová (2012) uvádza, že rozhodujúcim výrobným faktorom sa stávajú informácie a čas. Za úspešný bude možné považovať taký podnik, ktorý bude

mať svoje podnikateľské (riadiace, výkonné a správne) činnosti a procesy podporované informačnými technológiami, t.j. takými riadiacimi a výkonnými postupmi a metódami, ktoré vyžadujú počítačovú podporu na úrovni príslušnej doby. Súčasťou systému manažérstva kvality by mal byť preto kvalitný informačný systém (Kučera, Štefánek, Cvečko, 2002).

1.2.1 Vymedzenie základných pojmov v informačných systémoch

V kontexte s implementáciou a používaním informačných systémov sa často stretávame s termínmi dáta, informácie, prípadne znalosti.

Pojem *dáta* sa používa pre numerické, textové, obrazové a ďalšie údaje vtedy, keď nejde o to, aký vplyv má na entropiu príjemcu, ale ide iba o formu ich vyjadrenia, uloženia a spracovania (Bruckner a kol., 2012).

Dáta sú vhodným spôsobom zachytené správy, ktoré vypovedajú o svete a sú zrozumiteľné pre príjemcu, ktorým môže byť človek alebo technické zariadenie. Dáta vyžadujú na svoje spracovanie vynaloženie určitej práce, ktorá má zmysel iba vtedy, ak sa tým vytvorí istá úžitková hodnota. A je to práve informačný obsah, ktorý je úžitkovou hodnotou dát (Molnár, 2001).

V niektorých publikáciách sa môžeme okrem pojmu dáta stretnúť aj s pojmom *údaje*. Ako uvádza Hvorecký (2013), dáta a údaje sú synonymá. Označujú sa nimi akékoľvek hodnoty (fakty, zistenia, konštatovania, výsledky meraní a pozorovaní a pod.) zaznamenané v dohodnutej podobe. Využívajú sa ako podklad pre informovanie tých, ktorí sa o ne zaujímajú.

Informáciou rozumieme spracované údaje, ktorým ich používateľ prisudzuje určitý význam, a ktoré uspokojujú konkrétnu objektívnu informačnú potrebu svojho príjemcu. Informácie dávajú dátam význam a u príjemcu znižujú entropiu (neurčitosť) (Rábová, Drlík, 2013; Bruckner a kol., 2012).

Mariaš (2000) uvádza, že informácia je výsledkom interpretácie údajov na základe individuálnych schopností, hodnôt a vedomostí. Pritom vedomosti sú výsledkom aktívneho učenia sa. Vedomosti vymedzujú základné rámce na interpretáciu. Sú preto predpokladom využitia informačných systémov.

Informáciami označujeme závery zostavené na základe faktov, ktoré nás zaujímajú preto, lebo sa nimi chceme riadiť pri rozhodovaní. Rozdiel medzi údajom a informáciou spočíva teda predovšetkým vo význame zistenia pre konkrétnu osobu

v tom, či údaj (alebo záver vytvorený spracovaním skupiny údajov) považuje za natoľko dôležitý, že ovplyvní jeho rozhodovanie (Hvorecký, 2013).

Informácie sú teda z časti i zdrojom „mocenským“, lebo kto má potrebné informácie v predstihu pred ostatnými, môže tým získať aj výraznú konkurenčnú výhodu. V tomto zmysle, najmä ak niekto potrebné informácie nemá na požadovanom mieste a v požadovanom čase, môže sa napr. v podnikaní dostať do zložitých situácií. Hovoríme o kritických informačných potrebách, ktoré vymedzujú nevyhnutné potreby riadiaceho subjektu pre zaistenie jeho úspešnej činnosti. Kto včas vlastní potrebné informačné zdroje, ten si zároveň vytvára výhodu prístupu i k informáciám ďalším. Platí to nielen medzi jednotlivcami, ale i vo vzťahu kolektívov (organizácií) navzájom. Je však potrebné zdôrazniť, že nestačí len vlastniť informačné zdroje, ale toto vlastníctvo musí byť podporené schopnosťami údaje obsiahnuté v týchto zdrojoch transformovať na informácie. Informácie sú stavebnými kameňmi pre znalosti manažérov a ich spolupracovníkov a tak i jednou z kľúčových podmienok ich úspešnej práce (Kučera, Látečková, 2004).

Definíciu samotného informačného systému venujeme celú nasledujúcu podkapitolu.

1.2.2 Informačný systém

Predpokladom úspešnej práce manažérov dnešnej doby je schopnosť pracovať s rozsiahlymi objemami údajov, vyznať sa v nich, vedieť z nich odvodzovať relevantné závery a na ich základe ďalej rozhodovať. V tejto činnosti im pomáhajú informačné systémy, ktoré zásadne ovplyvňujú spôsob práce s údajmi a informáciami, ako aj spôsoby rozhodovania a komunikácie (Košovská, Váryová, Ferenczi Vaňová, 2013).

Systém chápeme ako množinu prvkov spolu s ich vlastnosťami a vzťahmi medzi nimi, ktoré vykazujú ako celok určité vlastnosti. Systém je množina vzájomne prepojených komponentov, ktoré musia pracovať dohromady pre celý systém tak, aby tento systém splnil daný účel (daný cieľ) (Klimeš, Drlík, 2007).

Informačný systém je komplex informácií, ľudí, použitých informačných technológií, organizácie práce, riadenia podniku a technických prostriedkov a metód slúžiacich k zberu, prenosu, uchovávaníu a ďalšiemu spracovaniu dát za účelom tvorby a prezentácie informácií. Metóda nám hovorí, čo je potrebné urobiť v určitej fáze vývoja alebo prevádzky (Rábová, Drlík, 2013).

Molnár (2001) v podobnej definícii uvádza, že informačný systém je usporiadaný súhrn prvkov a činností, spolu s ich vlastnosťami a vzťahmi, ktorý transformáciou dát vytvára informácie pre používateľa. Informačný systém teda predstavuje usporiadaný súbor ľudí, metód a technických prostriedkov zabezpečujúcich zber, prenos, uchovávanie a spracovanie dát za účelom tvorby a prezentácie informácií pre potreby používateľov činných v systémoch riadenia.

Mariaš (2000) tieto definície dopĺňa nasledujúcou: „Informačným systémom je formalizovaný súbor procedúr navrhnutých na premenu údajov tak z vnútorných, ako aj vonkajších zdrojov na informácie dostupné pri rozhodovaní a plánovaní.“

Informačný systém je vždy nejakým spôsobom organizovaný a začlenený do organizačnej štruktúry podniku, má určité ekonomické charakteristiky a musí byť určitým spôsobom riadený jednak v dobe jeho vývoja, ako aj v dobe jeho prevádzky.

Informačný systém sa skladá z nasledujúcich komponentov:

- technické prostriedky (hardvér) – počítačové systémy doplnené o periférne jednotky,
- programové prostriedky (softvér) – tvorené systémovými programami, ktoré riadia chod počítača, efektívnu prácu s údajmi, komunikáciu počítačového systému s reálnym svetom a aplikačnými programami,
- dátové zdroje – potrebujú ich programové prostriedky,
- organizačné prostriedky (orgware) – súbor nariadení a pravidiel, ktoré definujú prevádzkovanie a využívanie informačných systémov a informačných technológií,
- ľudská zložka (peopleware) – rieši otázky adaptácie a účinného fungovania človeka v počítačovom prostredí,
- reálny svet (informačné zdroje, legislatíva, normy) – okolie informačného systému (Rábová, Drlík, 2013).

Informačný systém organizácie je systém informačných technológií, údajov a ľudí, ktorého cieľom je efektívna podpora informačných a rozhodovacích procesov na všetkých úrovniach riadenia organizácie (Rábová, Drlík, 2013).

Informačný systém zohráva významnú úlohu pri tvorbe a distribúcii informácií. V každom podniku sa nahromadilo veľké množstvo údajov, ktoré sú uložené v produkčných systémoch, prípadne sa získavajú z externých údajových zdrojov. Takto získané údaje potom slúžia k riešeniu otázok kladených manažmentom podniku (Kučera, Látečková, 2004).

Akokoľvek informácie, ktoré pracovník používa pri svojej práci, môžeme z hľadiska informačného systému považovať za dáta. Činnosti, ktoré sú v biznise uskutočňované, sú v informačnom systéme, pokiaľ sú automatizované, označované ako funkcie a skupina takýchto činností je označená ako funkcionalita informačného systému. Pracovníci v biznise (aktéri procesov) sú z hľadiska informačného systému jeho používatelia. Skupiny činností, ktoré sú spoločné pre určité pracovné miesto sú v informačných systémoch označené ako role, ktoré je možné priradiť určitému typu používateľa (Bruckner a kol., 2012).

Pre informačný systém musí byť charakteristickou vlastnosťou usporiadanosť informácií. To umožňuje, aby informačný systém bol charakterizovaný týmito vlastnosťami:

- strategická orientácia – informačný systém musí byť schopný podporovať dosiahnutie strategických cieľov podniku,
- adekvátne funkčné spektrum – informačný systém musí pokrývať všetky požiadavky používateľov,
- relevantnosť – informačný systém umožňuje riešiť skutočné problémy pracoviska,
- dostupnosť – jednotliví používatelia majú možnosť prístupu k informáciám, ktoré skutočne potrebujú,
- prehľadnosť – možno sa rýchlo zorientovať v informáciách,
- presnosť a zrozumiteľnosť – jednoznačnosť chápania informácie,
- jednoduchá doplniteľnosť – a to nielen získaných informácií, ale aj softvérových riešení špecifických problémov, možnosť priebežnej aktualizácie údajov (Mariaš, 2000).

Aplikačný softvér alebo *počítačovú aplikáciu* definujeme ako softvér, ktorého cieľom je počítačová podpora častí informačných systémov. Často sa tento pojem zamieňa s celým integrovaným informačným systémom (Rábová, Drlík, 2013).

1.2.3 Systém pre plánovanie zdrojov (ERP)

Informačné systémy predstavujú v súčasnosti nevyhnutnú súčasť úspešného riadenia podniku. Ak hovoríme o informačných systémoch pre podporu podnikových procesov a pre podporu rozhodovania, myslíme tým podnikové aplikácie so súhrnným názvom *systémy pre plánovanie podnikových zdrojov – ERP (Enterprise Resource Planning)*. ERP je príkladom aplikačného softvéru (SAP, Helios, QI, Microsoft

Dynamics Nav). ERP sú celopodnikové aplikácie orientované na podporu biznis transakcií, resp. aplikácie pre plánovanie a riadenie podnikových zdrojov. Tieto aplikácie sa orientujú nielen na samotné plánovanie, ale dá sa povedať, že spravujú celý životný cyklus zdrojov podniku. Sú funkčným jadrom podnikovej informatiky, najmä u spoločností výrobného, obchodného alebo servisného charakteru. Jedná sa o integrovaný počítačový systém, ktorý sa používa na riadenie a plánovanie interných a externých zdrojov, vrátane hmotného majetku, finančných zdrojov, materiálov a ľudských zdrojov. ERP predstavuje softvérovú aplikáciu, ktorej cieľom je riadiť tok informácií naprieč všetkými procesmi v podniku a spravovať ich napojenie na externé procesy (Gála, Pour, Šedivá, 2015; Rábová, Drlík, 2013).

Košovská, Váryová, Ferenczi Vaňová (2013) definujú informačný systém kategórie ERP ako účinný nástroj, ktorý je schopný pokryť plánovanie a riadenie hlavných interných podnikových procesov a to na všetkých úrovniach, od operatívnej až po strategickú. ERP predstavuje balíkový podnikový programový systém, ktorý umožňuje automatizovať a integrovať väčšinu podnikových procesov, zdieľať podnikové údaje v rámci celého podniku. Kľúčovými internými procesmi, ktoré ERP systémy pokrývajú, sú výroba, vnútorná logistika, personalistika a ekonomika. Hlavným zmyslom týchto systémov je integrovať jednotlivé podnikové funkcie na úrovni celého podniku, teda integrovať rôzne, v podniku používané, aplikácie zabezpečujúce informačné potreby jednotlivých odborov a oddelení do jedného celku pracujúceho nad spoločnou dátovou základňou, a znížiť tak riziko neefektívnosti spracovania a vzniku mnohých chýb v podnikových dátach.

Podľa toho, ako ERP aplikácie pokrývajú všetky kľúčové oblasti podnikového riadenia a v závislosti na úrovni podpory integrácie podnikových procesov, ich môžeme deliť na:

- All-in-one – predstavujú rozsiahle aplikačné softvéry charakteristické rozsiahlou funkcionalitou a pokrývajú komplexne celé podnikové riadenie,
- Best-of-breed – špecializujú sa na vybranú podnikovú oblasť alebo na oblasti či procesy, ktoré sú špecifické iba pre podniky určitého odvetvia (napr. v automobilovom, chemickom priemysle, v maloobchode a pod.),
- Lite ERP – predstavujú odľahčené verzie ERP systémov určené predovšetkým pre malé a stredné podniky (Gála, Pour, Šedivá, 2015).

ERP má niekoľko charakteristických vlastností:

- schopnosť automatizovať a integrovať kľúčové podnikové procesy,

- zdieľať spoločné údaje a spracovávať ich v rámci celého podniku,
- vytvárať a sprístupňovať informácie v reálnom čase,
- spracovávať historické údaje (Rábová, Drlík, 2013).

ERP zabezpečujú širokú škálu funkcií a vo vzťahu k ostatným typom aplikácií sú z tohto pohľadu najkomplexnejšie. Jednotlivé funkcie sú združované do aplikačných modulov. Rozsah funkcií i rozsah aplikačných modulov môže byť u rôznych aplikácií ERP rôzne rozsiahly (Gála, Pour, Šedivá, 2015). Každý modul poskytuje rôzne formy údajov z rôznych oddelení v rámci podniku. Možnosťou je, aby podniky používali len niektoré časti ERP jednotky. Každý modul musí byť schopný pracovať samostatne, ale tiež integrovať s ostatnými modulmi systému. Tieto moduly sú dôležité, pretože umožňujú podniku, aby sa viac zefektívnil v oblasti výroby a spracovania objednávok (Filová, Mendrošová, 2009).

Najmodernejšie ERP systémy sa nezaoberajú len svojimi doménovými oblasťami, ktorými sú riadenie financií, distribúcie, ľudských zdrojov a výroby, ale aj riešeniami pre riadenie vzťahov so zákazníkmi (CRM), riadenie dodávateľského reťazca (SCM), elektronické obstarávanie, dátové sklady (DW) a rôzne formy e-business oblastí (Kováč, 2009).

Vo väčšine podnikov predstavuje ERP základnú podnikovú aplikáciu, na ktorú sa potom viažu všetky ďalšie. V prostredí výrobných podnikov sú aplikácie ERP veľmi často prepojené s aplikáciami riadenia výroby (strojov a zariadení), s aplikáciami automatizácie skladov a s aplikáciami riadenia životného cyklu produktu (Rábová, Drlík, 2013). Pre ostatné typy aplikácií je ERP zdrojom dát. Vytvára a udržiava tak základnú databázu podniku. Dáta sú poskytované ďalším aplikáciám podniku (riadenie vzťahov so zákazníkmi, riadenie dodávateľských reťazcov, ...). Z toho vyplýva, že pre kvalitu väčšiny ďalších aplikácií nadväzujúcich na ERP je významná kvalita dát a databáz, ktoré ERP vytvára, aktualizuje a spravuje. Inými slovami chyby v ERP a nízka kvalita jeho databáz znehodnocujú aj ďalšie aplikácie, ktoré by mali zvyšovať celkovú úroveň informačného systému a podnikového riadenia (Gála, Pour, Šedivá, 2015).

Z pohľadu použitia dát v ERP systémoch môžeme identifikovať päť základných skupín dát:

- číselníky položiek, stredísk, účtov, dodávateľov, ...,
- kmeňové údaje o výrobkoch, strojoch, dodávateľoch, zákazníkoch, ...,

- zákazkové údaje o zákazke pre konkrétneho zákazníka, termínoch, množstve, štruktúre, spôsobe spracovania výrobku,...,
- archívne údaje o už zrealizovaných zákazkách,
- parametre ako hodnoty pre nastavenie optimálneho fungovania organizácie a modulov, realizácie výpočtov, zobrazovaní údajov, nastaveniach tlače (Rábová, Drlík, 2013).

Účinný ERP systém by mal pomáhať zamestnancom pri ich práci a poskytovať relevantné podklady pri ich každodennom rozhodovaní. Mal by sa prispôbiť firemným procesom a roliam užívateľov a byť otvorený z hľadiska možnej integrácie s ostatnými systémami (Košovská, Váryová, Ferenczi Vaňová, 2013).

ERP svojou základnou funkcionalitou umožňujú používateľom:

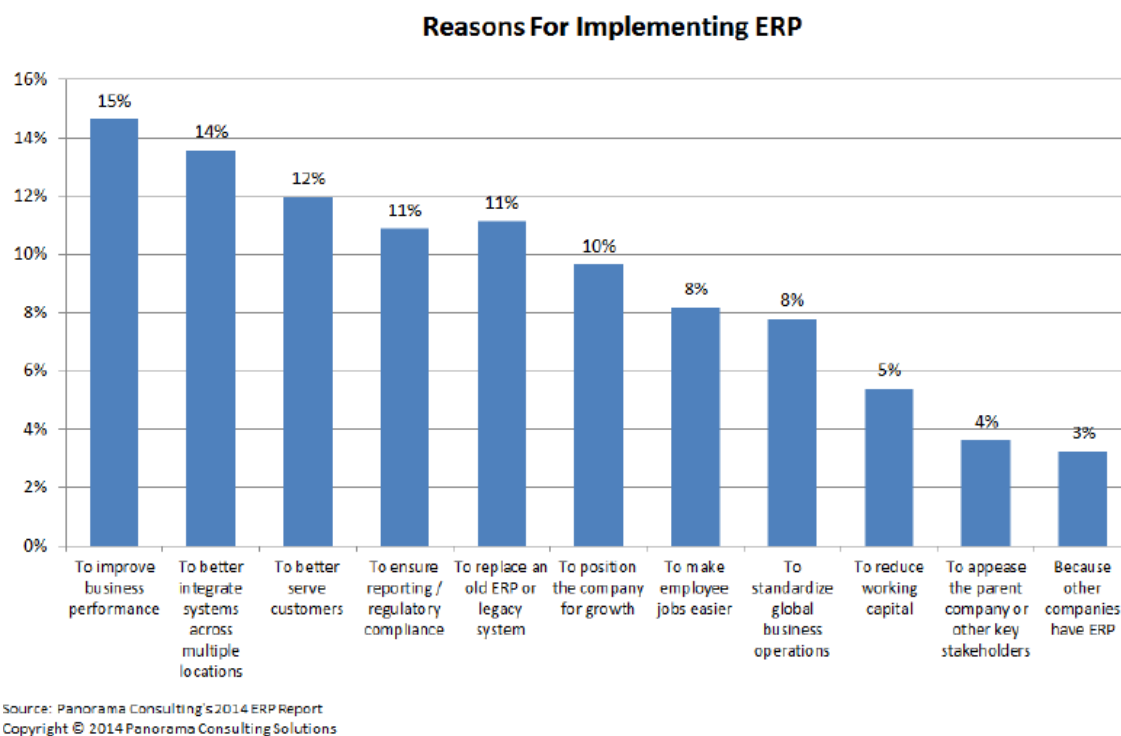
- vytvárať a aktualizovať rozsiahle dátové bázy – produkty, dodávateľov, zákazníkov, pracovníkov, majetky, účty a pod.,
- realizovať procesy operačného charakteru, t.j. spracovanie obchodných prípadov – nákupu materiálov, predaj produktov a ďalších a s tým súvisiacich obchodných dokumentov (objednávok, kontraktov, faktúr, colných deklarácií),
- vytvárať a prezentovať požadované prehľady, štatistiky a základné analýzy (prehľady zákazníkov, produktov, predaju, stavu zásob v sklade a pod.) (Gála, Pour, Šedivá, 2015).

Ambíciou ERP je pokryť celý systém kompletne integrovaným riešením, prípadne byť aspoň primárnym systémom medzi ostatnými aplikáciami. Funkcionalita ERP však obvykle neobsiahne všetky potreby podniku. Aj keby sa podarilo nejakým ERP nahradiť kompletne všetky existujúce aplikácie, nepodariť sa zrejme tento systém udržať ako jediný v budúcnosti, pretože technologicky zostarne a objavia sa ďalšie nové funkcionality a technologické príležitosti u iných softvérov (Bruckner a kol., 2012).

1.2.4 Výskum v oblasti implementácie a využívania ERP systémov

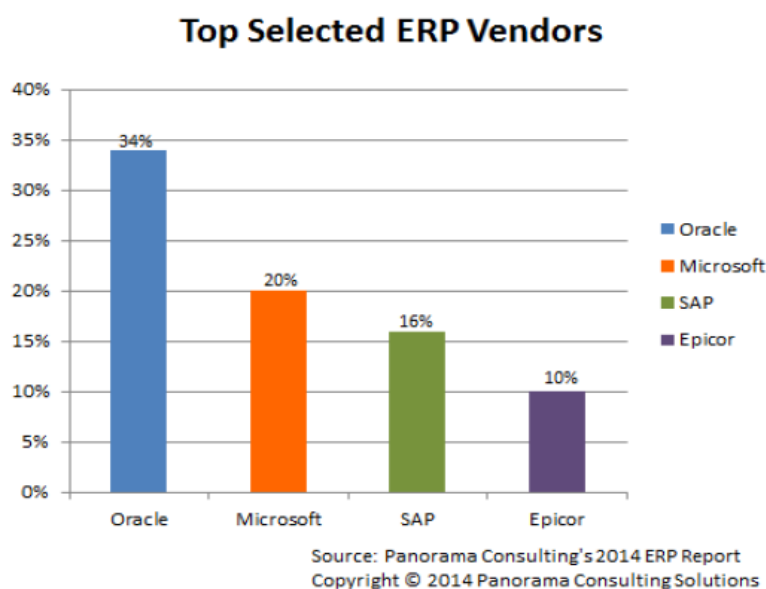
Poradenská spoločnosť Panorama Consulting Solutions realizovala v roku 2014 nezávislý výskum v oblasti implementácie a využívania ERP systémov, do ktorého sa zapojilo 192 podnikov.

Najčastejším dôvodom, ktorý podniky uviedli, pre zavedenie ERP bolo zlepšenie výkonnosti podniku (15%), zlepšenie integrácie systémov v rôznych lokalitách (14%) a zlepšenie služieb pre zákazníkov (12%). 3% podnikov uviedlo ako dôvod pre zavedenie ERP systému, že ERP systém majú ostatné podniky.



Graf 1 Dôvody pre zavedenie ERP systému¹

Pri výbere ERP systému si podniky najčastejšie do užšieho zoznamu vyberú systém od dodávateľa SAP, za ktorým nasledujú Oracle a Microsoft Dynamics. Z percentuálneho vyjadrenia výsledného výberu dodávateľa po tom, čo sa dostal do užšieho výberu, vyplýva, že podniky si najčastejšie zvolili Oracle (34%), následne Microsoft Dynamics (20%), SAP (16%) a Epicor (10%).



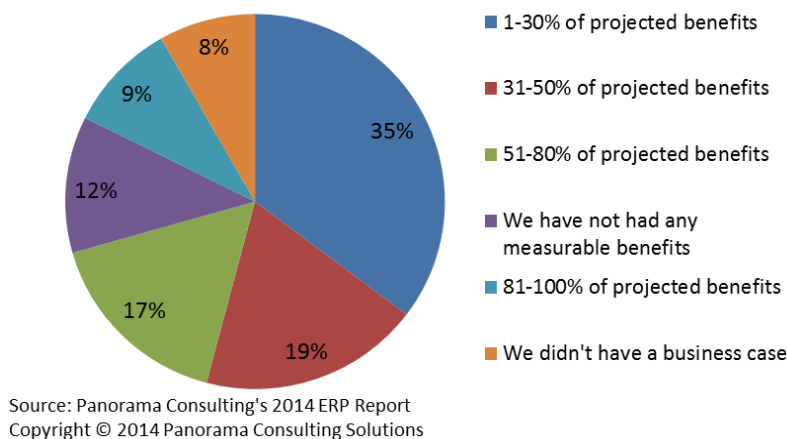
Graf 2 Najčastejšie vybraní ERP dodávateľia²

¹ Zdroj Graf 1: <https://partners.megaventory.com/assets/2014-ERP-Report-2.pdf>

² Zdroj Graf 2: <https://partners.megaventory.com/assets/2014-ERP-Report-2.pdf>

80% respondentov dosiahlo po zavedení ERP systému nejaké merateľné výhody. 66% respondentom priniesol ERP systém menej ako polovicu predpokladaných výhod (medzi 0 a 50 percentami). Iba 26% respondentom ERP systém priniesol 51 až 100% predpokladaných výhod.

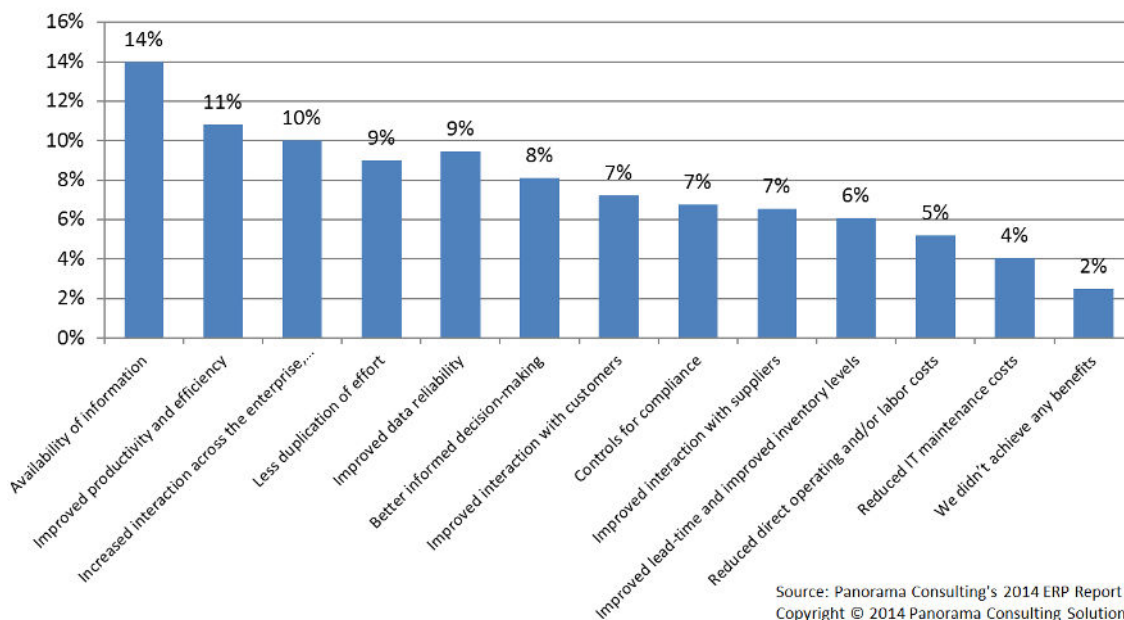
Percent of Benefits Realized



Graf 3 Percento získania predpokladaných výhod ERP systému³

Medzi najčastejšie získané výhody patrí lepšia dostupnosť informácií (14%), zvýšená produktivita a efektívnosť (11%), lepšia komunikácia v rámci firmy (10%), menej dvojitej práce (9%) a väčšia spoľahlivosť dát (9%).

Type of Benefits Realized



Graf 4 Získané výhody z ERP⁴

³ Zdroj Graf 3: <https://partners.megaventory.com/assets/2014-ERP-Report-2.pdf>

⁴ Zdroj Graf 4: <https://partners.megaventory.com/assets/2014-ERP-Report-2.pdf>

Skoro polovici respondentov (49%) sa návratnosť investícií do ERP systému preukázala do určitého časového obdobia, z toho v 17% prípadoch to bolo do dvoch rokov. Viac ako štvrtina respondentov (27%) tvrdí, že sa im nevrátil ich vklad do ERP systému a 24% respondentov to nijakým spôsobom nemeralo.

Zavedenie ERP systému sprevádzajú rôzne technické a prevádzkové úlohy, ktoré môžu spôsobiť prerušenie prevádzky. Podľa výskumu Panorama Consulting Solutions je to proces, ktorý predstavuje najväčšie riziko. Sice sa takéto komplikácie môžu vyskytnúť a vo vyše polovici prípadov (51%) sa aj vyskytli, ale zvyčajne sa jedná o krátkodobé prerušenie, ktoré sa dá rýchlejšie a ľahšie vyriešiť než rôzne organizačné úlohy.

1.2.5 Projektovanie podnikového informačného systému

Praktickými problémami vývoja rozsiahlych softvérových systémov sa zaoberá *softvérové inžinierstvo*. Zaoberá sa teda špecifikáciou, vývojom, riadením a údržbou softvérového produktu (Rábová, Drlík, 2013).

Softvérový produkt môžeme definovať ako ohraničenú množinu súvisiacich softvérových zložiek, ktoré sú vyvíjané s určitým cieľom a udržiavané na podnet používateľa (Rábová, Drlík, 2013).

Softvér v kontexte softvérového inžinierstva nezahŕňa len samotné programy, ale tiež súvisiacu dokumentáciu a konfiguračné dáta, ktoré sú potrebné k správne fungovaniu týchto programov. Profesionálne vyvíjaný softvérový systém sa často neobmedzuje na jediný program. Obvykle obsahuje viac samostatných programov a konfiguračných súborov, ktoré slúžia k správne nastaveniu týchto programov (Sommerville, 2011).

Projektovanie informačného systému zahŕňa analýzu a navrhovanie informačného systému, využitie CASE nástrojov a metódy a techniky riadenia projektov. Projektovanie sleduje krátkodobé ciele a venuje sa až podrobným funkciám a špecifikáciám všetkých aspektov systému (Kučera, Látečková, 2004).

CASE (*Computer Aided Software Engineering*) nástroje sú nástroje pre podporu analýzy a návrhu aplikácií. V súčasnosti všetky vo svete rozšírené objektovo orientované CASE nástroje vychádzajú z modelovacieho jazyka UML. CASE nástroje sú ideálnym nástrojom pre návrh informačných systémov, pretože umožňujú v rámci jediného prostredia identifikovať dôležité diagramy UML (ale aj procesné a dátové

modely) potrebné pre analýzu a návrh informačných systémov (Kanisová, Müller, 2004).

CASE nástroje umožňujú:

- modelovanie informačných systémov pomocou diagramov,
- generovanie zdrojového kódu,
- reverzné inžinierstvo,
- synchronizácia modelu a zdrojového kódu,
- tvorba dokumentácie (Rábová, Drlík, 2013).

Rábová a Drlík (2013) uvádzajú, že analýza je jedna z najvýznamnejších etáp životného cyklu vývoja softvérového produktu. Nasleduje po etape špecifikácia požiadaviek a vo všeobecnosti ju môžeme charakterizovať ako štúdium problému ešte skôr, ako by sme vyvinuli nejaké úsilie k jeho praktickej realizácii.

Tvorba informačného systému nie je iba automatizáciou poznaných biznis procesov. Výsledok analýzy je podkladom pre vylepšenie, optimalizáciu a úpravy biznis procesov tak, aby boli odstránené nájdené problémy a nedostatky a aby bol využitý potenciál podpory biznis procesov informačnými technológiami. Predpokladom pre takýto návrh je práve dostatočná analýza (Bruckner a kol., 2012).

Biznis analýza by mala identifikovať nejednoznačnosti v biznis systéme, odhaliť problémové miesta systému, zistiť, ktoré časti systému sú dlhodobo stabilné a ktoré sú dynamické a podliehajú častým zmenám. Pokiaľ by štruktúra informačného systému bola založená na charakteristikách, pri ktorých môžeme očakávať zmenu, bol by softvér po takejto zmene nepoužiteľný. Parametre, ktoré sa často menia, by mali byť ľahko meniteľné aj v informačnom systéme. Cieľom biznis analýzy je dostatočne podrobne z rôznych pohľadov poznať biznis procesy a popísať ich pomocou modelov a to tak, aby tieto modely mohli byť použité ako dostatočné zadanie pre tvorbu informačného systému. Biznis analýza môže byť tiež podkladom pre analýzu požiadaviek (Bruckner a kol., 2012).

Analýza systému je činnosť, ktorá rozkladá zložitý systém tak, aby bol lepšie pochopiteľný. Výsledkom analýzy systému môže byť okrem rôznych informácií o častiach systému tiež aj model systému. Modelom všeobecne rozumieme reprezentáciu niečoho iného, čo má podobné niektoré vlastnosti ako to pôvodné, čiže pomocou tejto reprezentácie je možné to pôvodné poznať. Model je zvyčajne zjednodušený. Sú v ňom teda zvolené len niektoré vlastnosti toho pôvodného podľa

zvoleného hľadiska. V reálnom svete to môže byť model auta ako hračka napodobňujúca skutočné auto, maketa stavby, atď. (Bruckner a kol., 2012).

Každý model prezentuje odlišný pohľad či perspektívu daného systému. Najdôležitejším aspektom modelu systému je to, že vynecháva podrobnosti. Model je abstrakcia študovaného systému, nie jeho alternatívna reprezentácia (Sommerville, 2011).

V prípade informačných systémov sa ich model vytvára pomocou diagramu. V rámci tejto diplomovej práce budeme používať modely definované v UML. Takéto modely reprezentujú daný informačný systém z určitého hľadiska (napr.: dátový model, kontextový model, prípadov použitia,...). Modelovací jazyk UML (*Unified Modeling Language*) je univerzálny jazyk pre vizuálne modelovanie systémov. Jazyk UML bol navrhnutý preto, aby spojil najlepšie existujúce postupy modelovacích techník a softvérového inžinierstva. Modely sú pochopiteľné aj pre zadávateľa aplikácie a umožňujú kvalitné vyjasnenie požiadaviek používateľov na vytváraný systém. UML je tiež jazyk pre vizualizáciu, špecifikáciu, stavbu a dokumentáciu softvérových systémov (Arlow, Neustadt, 2011; Kanisová, Müller, 2004).

1.2.6 Životný cyklus informačného systému

V softvérovom inžinierstve nazývame proces tvorby softvéru ako životný cyklus vývoja. Pri spracovaní tejto podkapitoly sme sa opierali najmä o odborné publikácie Bruckner a kol. (2012) a Rábová, Drlik (2013). Životný cyklus vývoja softvéru obsahuje tieto etapy:

- špecifikácia problému,
- globálna analýza a návrh,
- detailná analýza a návrh,
- implementácia,
- zavádzanie do prevádzky,
- prevádzka, údržba a rozvoj.

Tieto etapy sa nazývajú etapami životného cyklu softvéru, pretože softvér prechádza svojimi vývojovými fázami, ktoré sa obvykle po určitej dobe cyklicky opakujú. K návratu z poslednej etapy prevádzka, údržba a rozvoj do prvej etapy špecifikácia problému dochádza vtedy, keď nové požiadavky na príslušnú aplikáciu už nie sú realizovateľné jednoduchou údržbou, ale je potrebné urobiť zásadné inovácie informačného systému.

Účelom etapy **špecifikácie problému** je posúdiť realizovateľnosť projektu, spracovať návrh riešenia projektu, resp. navrhnuť alternatívne riešenia a vybrať z nich najvhodnejšie, zostaviť časový harmonogram realizácie, odhadnúť celkové náklady a prínos projektu. Súčasťou etapy špecifikácie problému je úvodná štúdia, ktorá predstavuje záverečný dokument etapy, obsahuje vybranú alternatívu riešenia a úplnú dokumentáciu pre vývoj systému. Vytvára sa aj špecifikácia požiadaviek a to vo forme formálnej alebo neformálnej špecifikácie. Neformálna špecifikácia je stručný popis vytváraného IS, prípadne popis problémovej domény. Formálna špecifikácia predstavuje už detailnejší popis požiadaviek, ktoré sú rozdelené na funkčné a nefunkčné požiadavky. Na výsledku etapy špecifikácia problému sa rozhoduje o realizácii projektu. V prípade, že sa príde k záveru, že projekt je príliš rozsiahly, je vhodné ho rozdeliť na menšie časti, ktoré budú riešené postupne prostredníctvom samostatných projektov. Ďalšie etapy vývoja potom prebiehajú pre jednotlivé subprojekty samostatne. V prípade, že sa príde k záveru, že požiadavky sú nerealizovateľné, životný cyklus daného softvéru končí a projekt je nutné prehodnotiť.

Globálna analýza a návrh vychádza z koncepcie stanovenej v úvodnej štúdii. Cieľom je spresnenie a rozpracovanie základných požiadaviek na systém, ktoré vznikli v predchádzajúcej etape. Podrobne sú popísané funkčné aj dátové požiadavky, známa je ich priorita, je vytvorená štruktúra systému. Vytvára sa konceptuálny návrh systému, ktorý rieši obsah a podstatu aplikácie pre biznis v pojmoch biznisu, teda nezávisle na implementačnom prostredí aplikácie i na technologickej platforme určenej pre budúcu prevádzku aplikácie.

Detailná analýza a návrh transformuje konceptuálnu úroveň návrhu do technologickej, ktorá je závislá na zvolenom type implementačnej a prevádzkovej platformy aplikácie. Cieľom fázy detailnej analýzy a návrhu je detailná špecifikácia na platformovo špecifickej úrovni. Zároveň sa navrhuje používateľské rozhranie. Vytvorený je úplný technologický model systému, ktorý vychádza z globálnej analýzy. Obsahuje detailný návrh architektúry systému, vrátane popisu rozhrania a používateľských obrazoviek, v dátovom modeli sú detailne navrhnuté štruktúry údajov (typy, dĺžky, presnosť, začiatkové a konečné hodnoty, ...).

Náplňou **implementácie** je transformácia technologickej úrovni návrhu IS do implementačnej úrovni, teda naprogramovanie programových modulov vo zvolenom vývojovom prostredí, realizácia fyzického návrhu databázy v konkrétnom databázovom prostredí, testovanie systému a kompletizácia dokumentácie. Neoddeliteľnou súčasťou

tejto etapy je príprava pre konverziu údajov alebo príprava materiálov, z ktorých sa údaje budú do databázy vkladať.

V rámci etapy **zavedenia do prevádzky** sa systém zavádza do prevádzky. Prebieha inštalácia technického a programového vybavenia. Postupuje sa podľa definovanej stratégie migrácie a vykonávajú sa odovzdávacie procedúry. Transformuje sa pôvodná dátová základňa predmetovej oblasti do stavu potrebného pre novú verziu informačného systému, školia sa používatelia aplikácie a realizuje sa skúšobná prevádzka aplikácie. Až po overení vhodnosti systému pre bežnú prevádzku a uskutočnenie všetkých schvaľovacích testov je táto etapa dokončená.

Prevádzka, údržba a rozvoj sú záverečnou fázou životného cyklu softvérového produktu. Softvérový produkt je v tejto fáze prevádzkovaný, používaný pre podporu biznis procesov a sú dosahované prínosy, pre ktoré bol projekt navrhnutý a v predchádzajúcich fázach realizovaný. Účelom prevádzky a údržby je v prvom rade zaistiť prevádzku systému, jeho údržbu a rozvoj vzhľadom k vývoju potrieb používateľov. Náplňou etapy je poskytovanie informatických služieb, organizačné, personálne, technické a materiálové zabezpečenie prevádzky, poskytovanie pomocných informácií a konzultácií, údržba systému a dokumentácie, ako aj riadenie zmien.

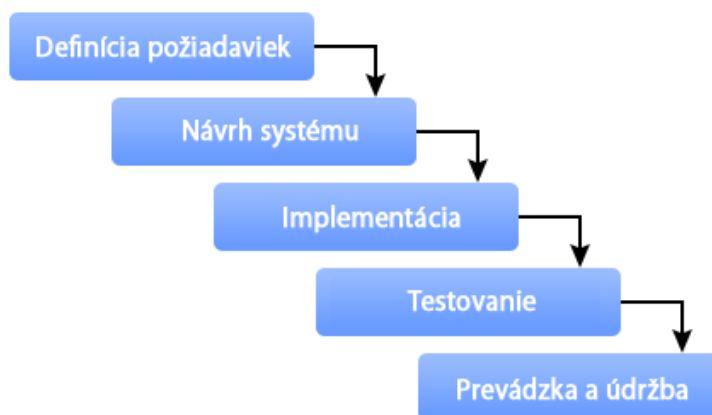
Ak informačný systém dospel do štádia, kedy už nie je schopný ďalšieho vývoja alebo je nevyhnutné uskutočniť zásadné zmeny v jeho štruktúre, je informačný systém na konci svojho životného cyklu. Pokiaľ už nie je potrebný, alebo je jeho prevádzka a údržba neefektívna vzhľadom k jeho prínosom, dochádza k jeho **vyraďeniu**. Vyradenie sa nedá uskutočniť vymazaním programov a dát aplikácie z prevádzkovej platformy. Je potrebné archivovať nevyhnutné dáta a súčasne je potrebné ošetriť tie väzby aplikácie a tie biznis procesy, ktoré boli vyradením dotknuté.

1.2.7 Modely životného cyklu informačných systémov

Model životného cyklu je rámec procesov a aktivít často organizovaných do jednotlivých fáz. Pri spracovaní tejto podkapitoly sme sa opierali najmä o odborné publikácie Sommerville (2011), Bruckner a kol. (2012) a Rábová, Drlík (2013). Historicky vzniklo niekoľko modelov životného cyklu, z ktorých sú najvýznamnejšie vodopádový, iteratívny a inkrementálny model.

Tieto modely sa vzájomne nevyklúčujú a môžu sa používať spoločne, najmä pri vývoji veľkých systémov. V prípade veľkých systémov je rozumné kombinovať hlavné výhody vodopádového modelu a modelu inkrementálneho vývoja.

Vodopádový model má v softvérovom inžinierstve zásadný význam, pretože predstavuje prvý a priekopnícky model, vďaka ktorému vznikol metodicky riadený vývoj softvéru. Dovtedy bol vývoj neriadený, dokonca často chaotický, spojený s vysokým rizikom neúspechu. Vodopádový model sa inšpiroval postupmi v priemysle a rozdelil vývoj softvéru do postupne vykonávaných fáz. Vzhľadom k tomu, že postupuje formou kaskády jednotlivých fáz, ktoré pripomínajú vodopád, označuje sa tento model ako vodopádový model životného cyklu.



Obrázok 1 Vodopádový model

Vodopádový model predstavoval v dobe svojho vzniku (70. a 80. roky 20. storočia) významný pokrok, pretože rozdelil celý proces vývoja do jednotlivých fáz a umožnil systematický a opakovaný postup vývoja. Vodopádový model predstavuje príklad plánovaného procesu. Typická je tvorba detailných plánov hneď v úvode. Dôsledkom je, že sa nepočíta s rizikom vzniku nečakaných udalostí, ktoré zmenia pôvodné požiadavky. Jednotlivé etapy vývoja nasledujú presne za sebou, bez možnosti návratu do predchádzajúcej etapy, s minimálnymi možnosťami opráv a spätnej väzby. Testovanie prebieha až na konci projektu. Vodopádový model prináša problémy v tých prípadoch, keď nie je možné špecifikovať všetky požiadavky na začiatku projektu, alebo je nutné robiť počas vývoja časté zmeny. Problémom vodopádového modelu je neskorá integrácia softvéru, ktorá sa robí až po naprogramovaní všetkých modulov.

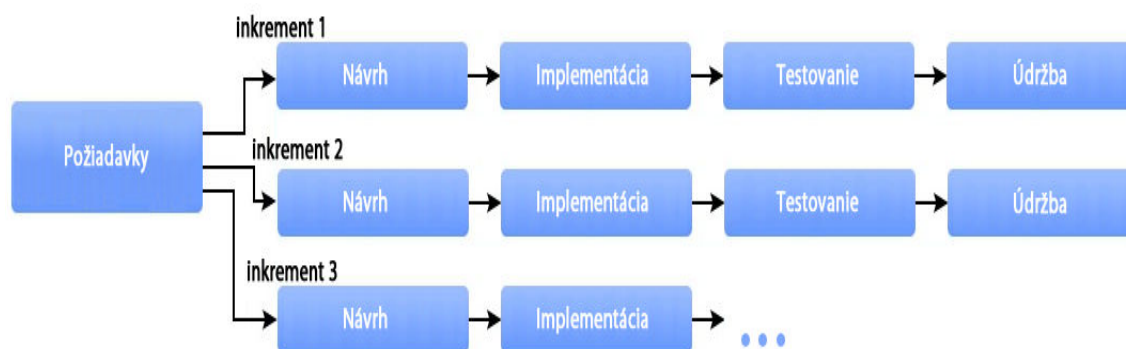
Nevýhody vodopádového vývoja odstraňuje **iteratívny vývoj**, ktorý je dnes moderným spôsobom vývoja. Iteratívny vývoj je postavený na skutočnosti, že človek lepšie rieši menšie problémy, a preto rozkladá celý projekt na rad čiastočných problémov – iterácií. Každá iterácia pritom obsahuje všetky etapy vývoja. Výsledkom každej iterácie musí byť fungujúca časť systému, ktorá je otestovaná a integrovaná s výsledkom predchádzajúcich iterácií.

Iteratívny vývoj vedie k podstatnému zníženiu rizík pri vývoji softvéru, pretože už prvá iterácia preverí prácu tímu vo všetkých fázach. Veľmi dôležitým prínosom je zmiernenie rizika nespokojnosti zákazníka, pretože tento spôsob vývoja zaisťuje rýchlu spätnú väzbu od zákazníka a všetkých zainteresovaných. Už po prvej iterácii zákazník vidí fungujúcu časť softvéru a môže naň reagovať. Každá iterácia má definovaný presný cieľ, ktorý sa snažíme naplniť. Iteratívny vývoj umožňuje lepšie sledovať postup projektu. Prípadné problémy sa odhalia včas a je ich tak možné lepšie riešiť. Zreťazením iterácií postupne rozširujeme jednotlivé funkčnosti softvéru až do výslednej podoby.

Iteratívny vývoj môže byť realizovaný dvomi základnými modelmi životného cyklu a to inkrementálnym alebo evolučným.

Inkrementálny vývoj, ktorý tvorí základnú zložku agilných prístupov, je pre väčšinu podnikov vhodnejší než vodopádový prístup. Inkrementálny vývoj prezentuje spôsob, akým ľudia bežne riešia problémy. Málokedy hneď vypracovávame úplné riešenie problému, ale postupujeme k nemu pomocou rady krokov. Keď si pri tom uvedomíme, že sme urobili chybu, vrátime sa o niekoľko krokov späť.

Inkrementálny model definuje na začiatku požiadavky na systém na hrubej úrovni. Potom sa systém rozdelí na samostatne realizovateľné časti (prírastky) a každý prírastok prechádza všetkými etapami procesu vývoja samostatne. Každý prírastok predstavuje časť plánovaného systému. Hlavná výhoda inkrementálneho vývoja spočíva v tom, že sa časti riešenia zavádzajú relatívne rovnomerne v priebehu projektu a zákazník sa tak môže skôr presvedčiť o tom, že sa riešenie vyvíja požadovaným smerom a môže poskytnúť rýchlejšiu spätnú väzbu riešiteľovi o kvalite riešenia (resp. jeho prírastku). Pri aplikácii inkrementálneho modelu je kľúčová otázka správneho určenia prírastkov tak, aby bolo možné vyvíjať prírastok samostatne bez závislosti na ešte nerealizovaných častiach.



Obrázok 2 Inkrementálny model

Evolučný model sa od inkrementálneho líši v tom, že všetky požiadavky nie sú vopred definované, ale definujú sa až na začiatku každej iterácie.

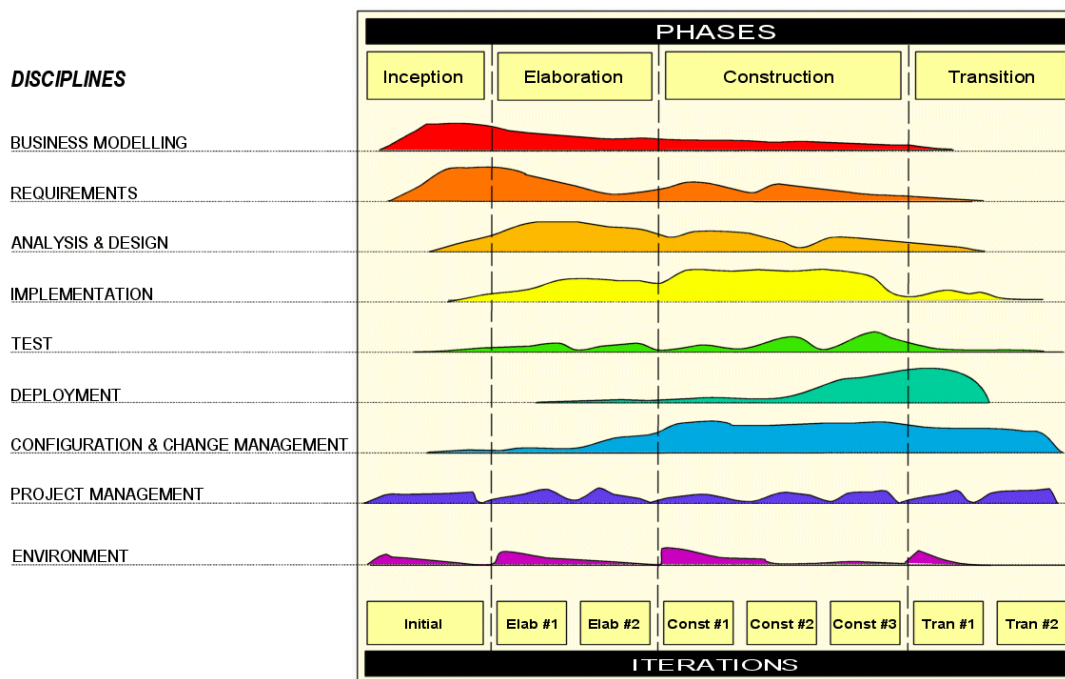
1.2.8 Metodiky vývoja softvéru

Pod metodikou vývoja informačných systémov budeme rozumieť súbor filozofií, fáz, procedúr, pravidiel, techník, pomôcok, dokumentácie a zásad riadenia projektov. Popisujú vytváranie informačných systémov s pevne stanovenými fázami a podfázami, ktoré odporúčajú techniky na použitie v každej fáze a podfáze, odporúčajú plánovanie, riadenie, sledovanie a vyhodnocovanie vyvíjaného produktu v jeho rôznych fázach (Kučera, Štefánek, Cvečko, 2002).

Metodika vývoja je odporúčaný súhrn princípov, konceptov, dokumentov, metód, techník a nástrojov pre tvorcov informačných systémov, ktorý pokrýva celý životný cyklus informačných systémov. Metodika určuje kto, kedy, čo, ako a prečo má robiť počas vývoja a prevádzky informačného systému. Cieľom metodiky je formalizovať postupy, definovať zodpovednosti a pravidlá komunikácie. Príkladom metodík informačných systémov sú SSADM, Oracle CASE Method, SDM, MMDIS, OMT, Booch, OOSE alebo UP a RUP (Rábová, Drlík, 2013).

Rational Unified Process (RUP) je založený na iteratívnom vývoji, ktorý je rozdelený na viacero fáz. V každej fáze prebieha jedna alebo viacero iterácií, pričom v iterácii sa vykonajú všetky disciplíny. Disciplíny sú súborom úloh a činností týkajúcich sa konkrétnej oblasti záujmu. Disciplínami sú biznis modelovanie, špecifikácia požiadaviek, analýza a návrh, implementácia, testovanie, projektové riadenie, správa konfigurácií, prostredie. Každá fáza ma definovanú množinu cieľov, ktorých splnenie sa overuje na konci fázy pri dosiahnutí tzv. míľnika. Pre každý míľnik sú špecifikované kritéria splnenia cieľa fázy, na základe ktorých sa rozhoduje o postupe do ďalšej fázy. RUP definuje 4 fázy:

- zahájenie (inception),
- príprava (elaboration),
- konštrukcia (construction),
- prechod (transition) (Rábová, Drlík, 2013; Bruckner a kol., 2012).



Obrázok 3 Fázy a disciplíny v RUP⁵

Cieľom fázy zahájenia je vytvorenie podnikového prípadu príslušného systému. Je potrebné identifikovať všetky externé entity (osoby a systémy), ktoré budú so systémom v interakcii. Na základe týchto informácií môžeme potom vyhodnotiť príspevok systému k fungovaniu podniku. Pokiaľ tento príspevok nie je významný, môže byť projekt zrušený. Fáza prípravy by mala rozvinúť znalosti problémovej domény, vytvoriť schému architektúry systému, vyvinúť projektový plán a identifikovať kľúčové projektové riziká. Fáza konštrukcie zahŕňa návrh systému, programovanie a testovanie. V tejto fáze sa paralelne vyvíjajú a integrujú časti systému. Na konci tejto fázy by mal byť k dispozícii funkčný softvérový systém a súvisiace dokumentácie. Záverečná fáza RUP prechod rieši prenesenie systému od vývojára k používateľovi a sprevádzkovanie systému v reálnom prostredí (Sommerville, 2011).

V súčasnej dobe sme svedkami rozmachu **agilných metodík** vývoja softvéru. Označenie agilný dostali tieto metodiky preto, lebo presne vystihuje ich silnú stránku – flexibilný, prispôsobivý, živý prístup k vývoju softvérového produktu so schopnosťou reagovať na zmeny, ktoré sa počas vývoja určite vyskytnú (Rábová, Drlík, 2013).

V rýchlo sa meniacom podnikovom prostredí majú preto najmä agilné metodiky kľúčovú rolu v prípade vývoja podnikových systémov. Tieto metodiky boli navrhnuté

⁵ Zdroj Obrázok 3: <http://projects.staffs.ac.uk/suniwe/images/RUPSummaryDiag.gif>

tak, aby dokázali rýchlo vyprodukovať užitočný softvér. Softvér sa nevyvíja ako jediná jednotka, ale v rade inkrementov, kde každý inkrement obsahuje nové funkcie systému. Agilné metodiky sú teda charakterizované inkrementálnym vývojom. Inkrementy sa sprístupňujú zákazníkovi každé dva či tri týždne. Agilné metodiky zapájajú zákazníkov do rozhodovacieho procesu, aby bolo možné získať rýchlu spätnú väzbu týkajúcu sa premenlivých požiadaviek (Sommerville, 2011).

Cieľom agilných metodík je zabezpečiť vytvorenie softvérového produktu rýchlejšie a efektívnejšie a ľahšie tak naplniť očakávania dnešných zákazníkov. Každá z agilných metodík je svojím spôsobom špecifická, ale všetky sú postavené na rovnakých princípoch a hodnotách agilného manifestu vývoja:

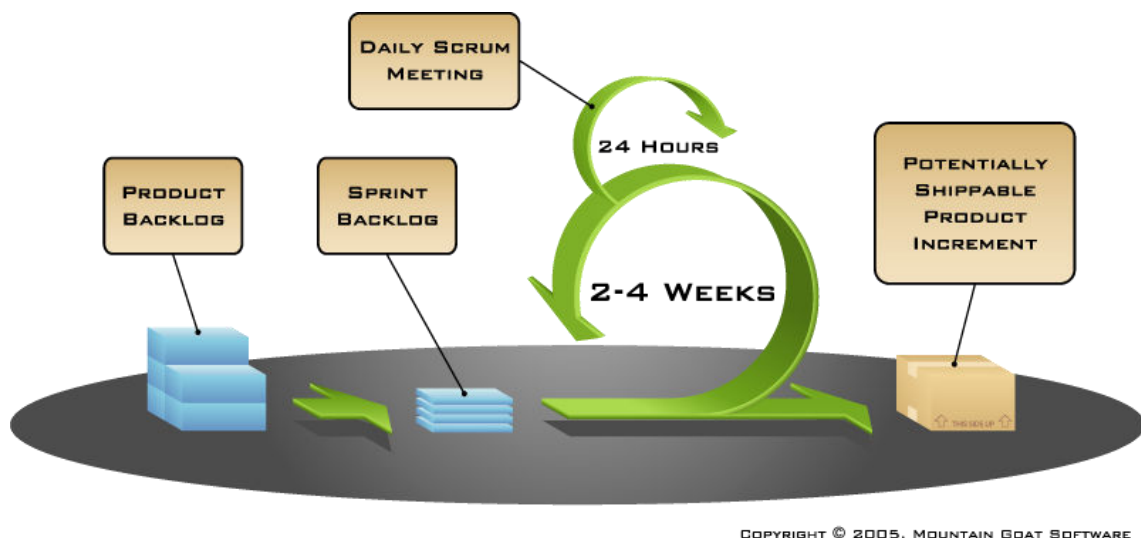
1. Najvyššou prioritou je uspokojovať zákazníka skorým a sústavným dodávaním hodnotného softvéru.
2. Zmeny požiadaviek sú vítané dokonca aj v neskorších fázach vývoja, keďže zmena môže zákazníkovi poskytnúť konkurenčnú výhodu.
3. Fungujúci softvér je potrebné dodávať často (od niekoľko týždňov do niekoľko mesiacov) s uprednostnením čo najkratších intervalov.
4. Ľudia z biznisu a vývojári majú denne spolupracovať počas celého projektu.
5. Kľúčovým faktorom úspechu projektu sú motivovaní jedinci, ktorí majú vytvorené podmienky pre prácu a majú podporu vedenia.
6. Najefektívnejším spôsobom prenosu informácií v tíme je osobná komunikácia.
7. Primárnym meradlom úspechu je fungujúci softvér.
8. Agilné procesy podporujú trvalo udržateľný vývoj. Sponzori, vývojári a používatelia by mali byť schopní udržať konštantné tempo rozvoja.
9. Neustále je potrebné sa zameriavať na perfektné technické riešenie a návrh, ktoré posilňujú agilitu.
10. Jednoduchosť – umenie vykonať naozaj len to potrebné je nevyhnutnosť.
11. Samoorganizované tímy vytvárajú najlepšie koncepty, požiadavky a návrhy riešení.
12. Tím v pravidelných intervaloch vyhodnocuje sám seba s cieľom byť efektívnejší a prispôbuje tomu svoje správanie (agilemanifesto.org, 2001).

Vo všeobecnosti sa týmito princípmi riadi skupina moderných metodík, ktoré vychádzajú z toho, že jedinou cestou, ako overiť správnosť navrhnutého systému je jeho rýchly návrh a predloženie zákazníkovi na pripomienkovanie (Rábová, Drlík, 2013).

Medzi agilné metodiky patria:

- Dynamic Systems Development Method (DSDM),
- Adaptiv Software Development (ASD),
- Feature Driven Development (FDD),
- Extrémne programovanie (Extreme Programming, XP),
- Lean Development,
- Scrum,
- Crystal metodiky,
- Agilné modelovanie (Agile Modeling) (Bruckner a kol., 2012).

Najpoužívanejšou agilnou metodikou je **Scrum**. Túto metodiku sme použili pri vývoji softvéru aj my. Metodika Scrum je zameraná hlavne na riadenie projektu. Postup vývoja v metodike Scrum je zachytený na nasledujúcom obrázku.



Obrázok 4 Postup vývoja v agilnej metodike Scrum⁶

Scrum je výhodný pre projekty s rýchlo sa meniacimi alebo vysoko naliehavými požiadavkami. V metodike Scrum vystupujú tri role: Product Owner, Scrum Master, Scrum Team. Product Owner reprezentuje všetky zainteresované strany a aj prvotné požiadavky nachádzajúce sa v Product Backlog. Scrum Master zodpovedá za Scrum Team a za priebeh projektu. Scrum Team pracuje na vývoji softvéru. Vývoj prebieha v iteráciách nazývaných Sprint. Sprint trvá zvyčajne 2 až 4 týždne. Zoznam požiadaviek, ktoré sú požadované zadávateľom, je definovaný v tzv. Product Backlog. Na začiatku každého sprintu sa koná Sprint Planning, počas ktorého Product Owner

⁶ Zdroj Obrázok 4: <http://epf.eclipse.org/wikis/scrum/Scrum/guidances/supportingmaterials/resources/ScrumLargeLabelled.png>

vyberie podľa priority požiadavky z Product Backlogu a tieto sú presunuté z Product Backlogu do Sprint Backlogu. Scrum Team si následne rozdelí úlohy, ktoré môžu urobiť v danom Sprinte na základe Sprint Backlogu. Každý deň sa koná krátke stretnutie s názvom Daily Scrum Meeting, ktoré riadi Scrum Master a zúčastňujú sa ich všetci členovia tímu. Na denných schôdkach sa rieši priebeh projektu, a tak sa aj monitoruje jeho stav. Na konci každého Sprintu sa koná Sprint Review Meeting, kde tím demonštruje dokončenú funkcionálnosť (eclipse.org, 2008).

2 CIELE DIPLOMOVEJ PRÁCE

Cieľom diplomovej práce je navrhnuť a vytvoriť podnikový informačný systém pre manažérstvo kvality výroby v podnikoch konkrétnej spoločnosti.

Podciele:

- analyzovať manažérstvo kvality, jej zabezpečovanie v podnikoch a využívanie podnikových informačných systémov,
- analyzovať aktuálny stav v oblasti manažérstva kvality v prostredí konkrétneho podniku vybranej spoločnosti,
- navrhnuť softvérové riešenie pre podporu procesov podnikového oddelenia zabezpečovania kvality,
- riešenie implementovať s využitím platformy Java EE a iných potrebných technológií so zameraním sa najmä na backendovú časť,
- testovať vyvíjaný informačný systém,
- na základe vykonaných testov zhodnotiť možnosti nasadenia implementovaného riešenia v praxi.

3 VÝVOJ PODNIKOVÉHO INFORMAČNÉHO SYSTÉMU

V tejto kapitole budeme pokladať informačný systém za softvérový produkt. Zameriame sa na celkový vývoj aplikačného softvéru informačného systému a to najmä jeho backendovej časti. Informačný systém je vyvíjaný pre účely manažérstva kvality v podnikoch vybranej nemeckej spoločnosti odkiaľ prišla požiadavka pre lepšie riadenie, organizáciu a plánovanie procesov týkajúcich sa kvality výroby. Pri spracovaní tejto časti sme sa opierali najmä o odborné zdroje Goncalves (2013), Jendrock a kol. (2014), Mukhar a kol. (2006), Vasiliev (2008), dokumentáciu (oracle.com, 2015) a vlastné praktické skúsenosti.

Za metodiku vývoja sme si zvolili Scrum. Rozhodli sme sa tak preto, lebo nám umožňuje dostatočne agilný vývoj a postupne vyvíjané inkrementy systému môžeme prebrať s vedením oddelenia zabezpečovania kvality a overiť si tak ich správnosť vzhľadom na požiadavky. Takýmto spôsobom zároveň môžeme podstatne rýchlo dodávať systém aj v jeho dočasných verziách, a tak pracovníci oddelenia zabezpečovania kvality ho môžu začať pomerne skoro používať a nemusia čakať až na jeho úplné dokončenie. Tiež sme predpokladali, že sa požiadavky na systém budú možno meniť aj počas vývoja, s čím taktiež táto metodika počíta.

V rámci projektového manažmentu sme používali Redmine. Za verzionovací systém sme si zvolili GitBlit. Vývoj prebiehal vo vývojovom prostredí Eclipse Luna 4.4.0 a aplikácia bola nasadzovaná a spúšťaná na aplikačnom serveri WildFly 8.2.

3.1 MANAŽÉRSTVO KVALITY VO VYBRANOM PODNIKU (PRÍPADOVÁ ŠTÚDIA)

Aby sme vedeniu oddelenia zabezpečovania kvality a aj pracovníkom tohto oddelenia mohli sprehľadniť prácu s dátami, dokumentmi, súbormi, zobrazit' štatistiky, pomôcť pri organizácii, či zautomatizovať a zefektívniť ich pracovné procesy, museli sme najskôr spoznať a pochopiť súčasné fungovanie procesov a jednotlivých činností pracovníkov v danom podniku. Keďže to nie je triviálna záležitosť, tak tento proces nášho zaúčania sa do problematiky manažérstva kvality prebiehal nielen na začiatku vývoja, ale aj počas celého vývoja. Celkový prehľad o priebehu procesov sme si vytvorili počas biznis analýzy a počas vývoja sme sa ešte viac zamerali na aktuálne vyvíjanú časť systému a v súvislosti s ňou bližšie špecifikovali a analyzovali aj práve riešenú časť pracovných procesov na danom oddelení.

3.1.1 Vytvorenie doménového slovníka

V tejto fáze vývoja bola dôležitá komunikácia s vedením oddelenia zabezpečovania kvality. Zo začiatku komunikácia prebiehala s vedením zo slovenskej pobočky, neskôr viacej s hlavným vedením z materskej pobočky v Nemecku. Pri komunikácii sa používa odborná terminológia špecifická pre danú doménovú oblasť a oddelenie. Významy jednotlivých termínov sme sa samozrejme taktiež museli naučiť, aby pri komunikácii nedochádzalo k nedorozumeniam a nešírili sa dezinformácie. Keďže sa jedná o medzinárodnú spoločnosť, termíny sme používali v troch jazykoch – slovenský, anglický, nemecký. Pre naše účely v rámci vývoja sme si vytvorili nasledovný slovník domény zobrazený v Tabuľke 1.

Tabuľka 1 Doménový slovník

EN	DE	SK
Production order		Zákazka.
Customer	Auftraggeber	Interný alebo externý zákazník.
Product	Material	Virtuálny produkt.
Article		Reálne vyprodukovaný výrobok.
Drawing	Zeichnung	CAD výkres.
Protocol		Protokol ako výstup činnosti hodnotenia kvality (merací p. alebo osvedčenie).
	Fertigungsauftrag	Dokument špecifikujúci detaily zákazky.
Order id	Auftragsnummer	Identifikačné číslo pre Fertigungsauftrag.
Internal order id	KundenAuftrag	Interné číslo zákazky.
Customer order id	Bestellnummer	Externé (zákazníkov) číslo zákazky.
Order date	Bestelldatum	Dátum vytvorenia zákazky v SAP ERP.
Order type	AuftragsArt	Kód reprezentujúci typ zákazky – či sa jedná o zákazku alebo reklamáciu.
Amount	Sollmenge	Množstvo výrobkov na výrobu.
Factory	Werk	Pobočka, kde sa zákazka spracováva.
Drawing index	Indexstand	Číslo verzie výkresu.
Product name	Materialkurztext	Názov výrobku.
Product type	AusgewahltesMaterial	Typ výrobku (interný/externý).
Article size		Výrobok je definovaný ich súborom.
Nominal size		Konkrétna hodnota pre article size.
Group of sizes		Sizes, ktoré môžu byť spracované rovnakým nástrojom.

Drawing tolerance		Minimálna a maximálna tolerancia pre každý article size.
Frame shift factor		Definuje, v ktorom ďalšom výrobku sa prejaví zmena, ak produkcia spraví korekciu v akomkoľvek rozmere.
TOP tolerance		Používa sa pri sériovej výrobe pre určenie kvality celej série. Môže byť definovaná percentuálne alebo počtom výrobkov.
Articles in OK quality		Výrobky spĺňajúce drawing tolerance.
Articles in TOP OK quality		Výrobky spĺňajúce TOP tolerance.
Articles in NOT OK quality		Výrobky nespĺňajúce drawing tolerance.

3.1.2 SAP ERP

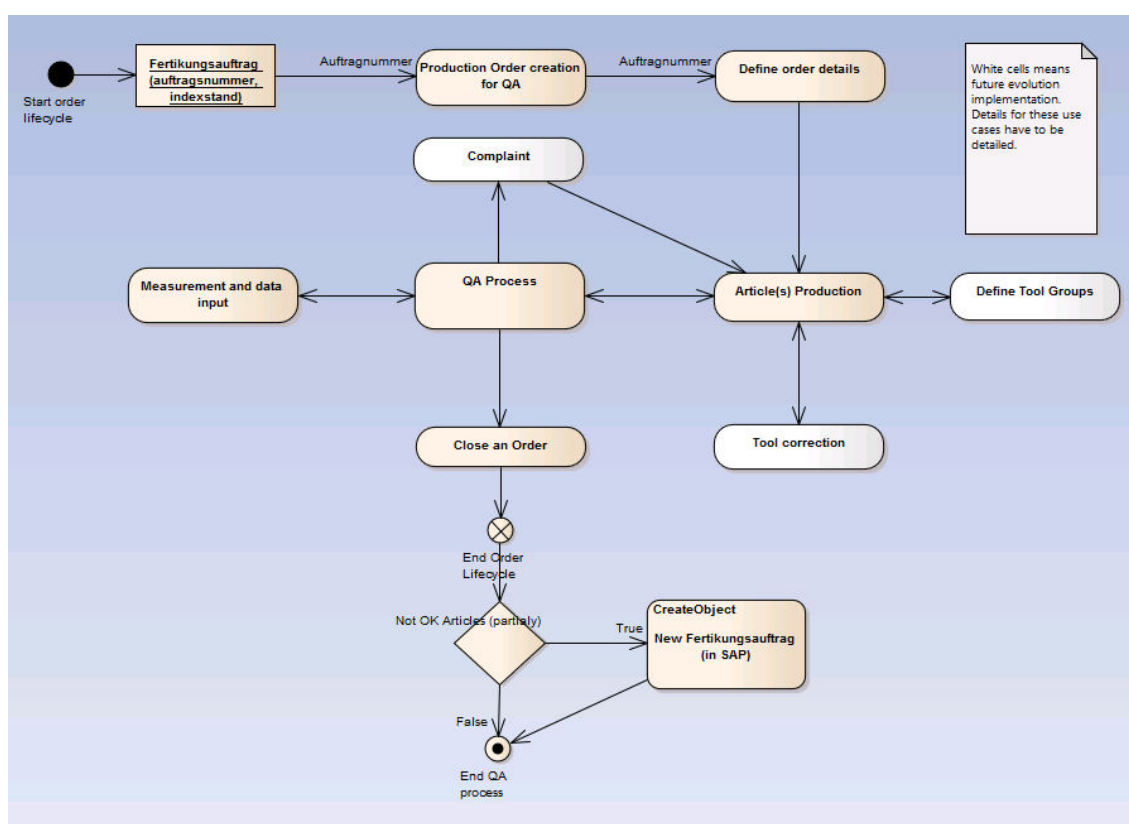
Mnohé používané termíny vychádzajú z pomenovaní, ktoré sú zároveň použité v systéme SAP ECC. Často sa môžeme stretnúť len s pomenovaním SAP, čo je neformálne označenie podnikového ERP systému dodávaného spoločnosťou SAP AG so sídlom v nemeckom Waldorfe. ERP systém spoločnosti SAP AG sa v jednej z prvých generácií označoval ako R/2. Jeho ďalšia generácia bola označená ako R/3 a s príchodom a rozvojom ďalších produktov spoločnosťou SAP AG bol tento produkt v poslednej generácii pomenovaný ako ERP alebo tiež ECC (Enterprise Central Component) (Jarunek, 2014).

SAP ERP je zároveň hlavným celopodnikovým systémom používaným v nami vybranej spoločnosti a jej podnikoch. Združuje dáta o všetkých zákazkách, produktoch, zákazníkoch, výkresoch a iné. Práve tieto dáta nás budú pri vývoji nášho softvéru zaujímať.

3.1.3 Proces výroby a kontroly kvality

Začiatkom celého procesu je prijatie zákazky od zákazníka, alebo určenie novej série výroby na základe sťažnosti z oddelenia zabezpečovania kvality (náprava chybných výrobkov). Zákazka je reprezentovaná dokumentom Fertigungsauftrag. Každý takýto dokument jednoznačne určujú dva údaje a to identifikačné číslo (auftragsnummer) a verzia výkresu (indexstand). Výkres popisuje produkt, ktorý je požadovaný v rámci zákazky. Na základe týchto dvoch údajov sa môže vytvoriť zákazka v nami vyvíjanom podnikovom informačnom systéme *Quality Assurance Management* (PIS QAM). Následne sa zadávajú detaily zákazky, čo znamená, že pracovník definuje aké merania majú byť uskutočnené a aj hodnoty tolerancií. V prvej

verzii vyvíjaného informačného systému sa vyžaduje, aby používateľ manuálne zadával údaje, neskôr je to riešené automatizovaným získavaním údajov zo SAP ERP alebo zo súborov programu Yocton. V ďalšom kroku prebieha produkcia výrobkov, ktorá je uskutočnená mimo oddelenia zabezpečovania kvality. Po ukončení výroby začína proces kontroly kvality. Spôsoby kontroly kvality sme popísali v prvej kapitole tejto diplomovej práce. Ak z kontroly kvality vyplynulo, že kontrolované výrobky spĺňajú zadané tolerancie, zákazka môže byť na oddelení zabezpečovania kvality uzavretá a výrobky môžu byť doručené zákazníčkovi. Ak sa ale zistilo, že výrobky nespĺňajú požadovanú kvalitu, musí byť vytvorená nová zákazka a cyklus sa opakuje. Celý cyklus je znázornený na obrázku 5.



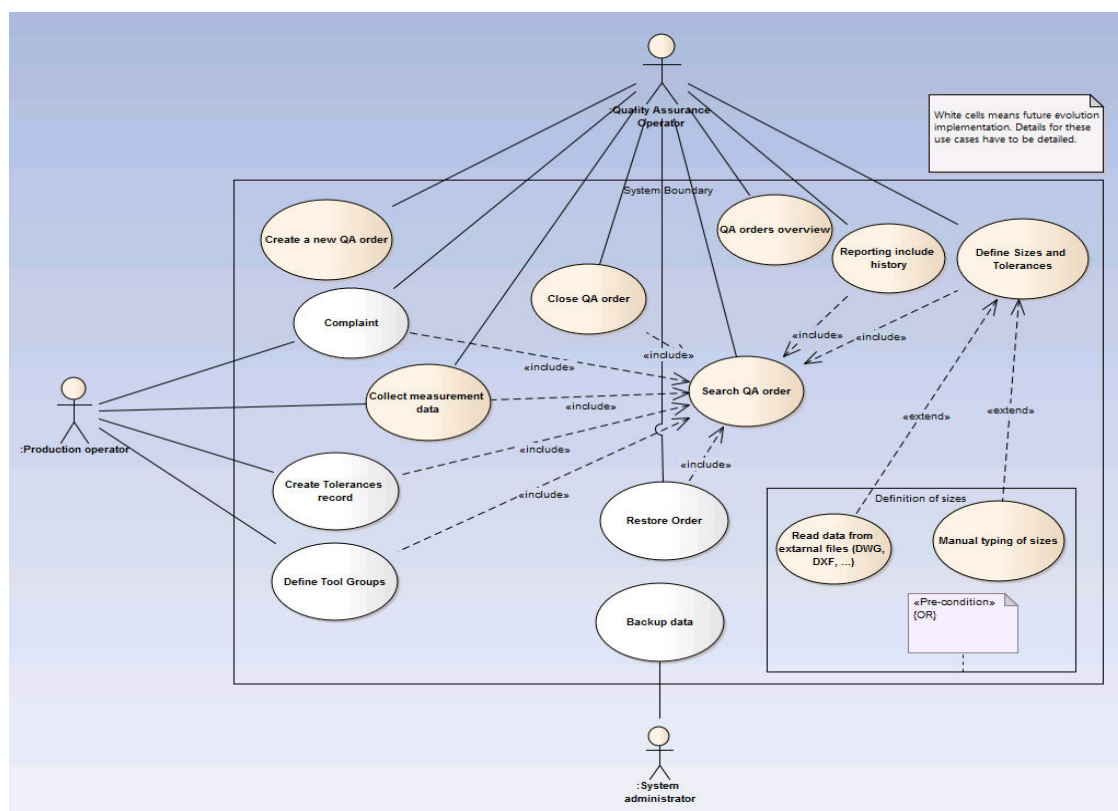
Obrázok 5 Proces výroby a kontroly kvality

3.2 ŠPECIFIKÁCIA POŽIADAVIEK

Hlavnou úlohou PIS QAM je podpora oddelenia zabezpečovania kvality podniku v organizovaní, zefektívnení a plánovaní ich pracovných procesov spojených s kontrolou kvality výroby. Cieľom je čo do najväčšej miery zautomatizovať procesy manažérstva kvality na tomto oddelení.

Jednotlivé požiadavky sme rozdelili do viacerých sprintov, v rámci ktorých boli požiadavky vypracované a dodávané ako verzie celého systému. Rozdelenie požiadaviek prebiehalo podľa ich rozdielnej priority. Priorita požiadaviek bola určená aj na základe žiadostí vedenia oddelenia. Požiadavky sa usporiadali podľa toho, čo je pre oddelenie zabezpečovania kvality najdôležitejšie a tiež najnevyhnutnejšie, aby mohli aj prvú verziu PIS QAM zapojiť do ich pracovných procesov a začať ju používať.

Prvá verzia PIS QAM je zameraná na základnú funkcionálnu. Má umožňovať manuálne vytvorenie zákazky v systéme, zobrazenie jej detailov, editáciu jej údajov, zobrazenie prehľadu všetkých zákaziek, uzatvoriť zákazku, zobrazenie prehľadu všetkých uzatvorených zákaziek, určenie potrebných protokolov pre konkrétnu zákazku a zobrazenie základných štatistík o zákazkách. Zároveň je potrebné brať do úvahy, že systém má byť prevádzkovaný v prostredí intranetu. Druhá verzia sa zameriava na protokoly. Používateľovi musí byť umožnené nielen vybrať potrebné protokoly pre danú zákazku, ale systém zvládne aj protokoly vytvárať. V tretej verzii sa v PIS QAM implementuje prepojenie na SAP ERP a systém má automatizovane z neho získavať dáta. Štvrtá verzia má byť integrovaná s meracími nástrojmi používanými pri kontrole kvality výrobkov. Piata verzia implementuje používateľský manažment a zvyšné funkcionality zobrazené bielou farbou na obrázku 6.



Obrázok 6 Diagram prípadov použitia

Na diagrame prípadov použitia môžeme vidieť, že sme si určili tri typy používateľov PIS QAM s rôznymi používateľskými právami:

- Pracovník oddelenia zabezpečovania kvality – bude mať prístup k všetkým funkcionalitám systému, pre ktoré bol systém navrhnutý.
- Pracovník výroby – bude mať prístup iba k jednej časti systému a to ku korekcii a konfigurácii nástrojov pri výrobnom procese.
- Administrátor – zabezpečuje zálohovanie dát na základe jeho konfigurácie.

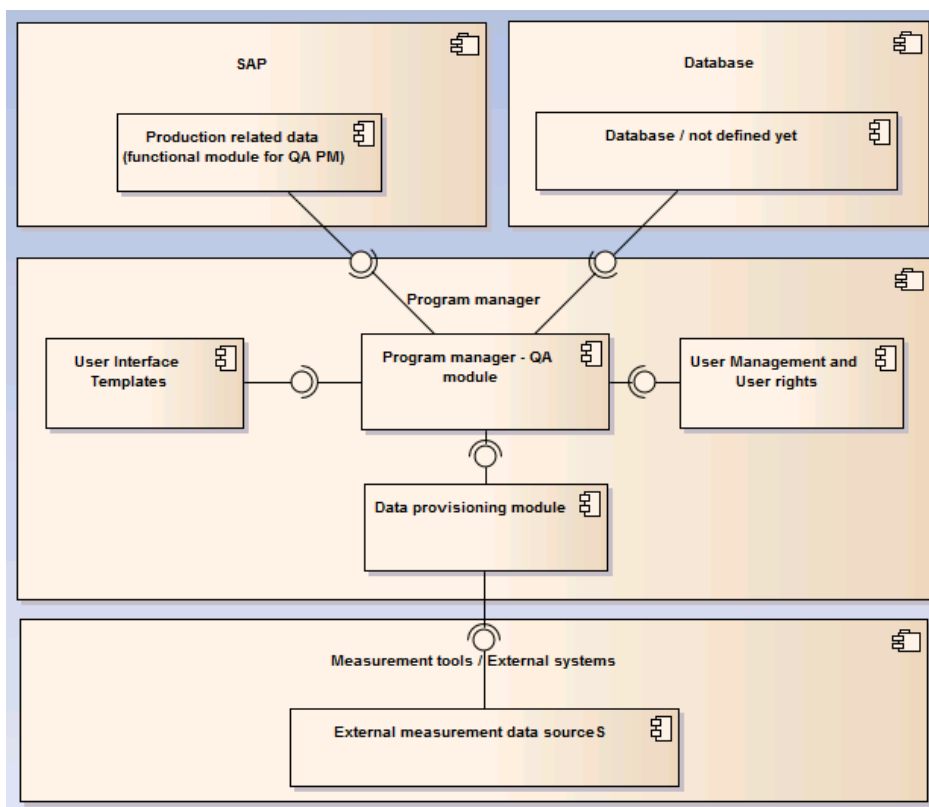
3.3 NÁVRH PIS QAM

Pôvodný návrh počítal s vývojom PIS QAM vo forme modulu, ktorý by bol súčasťou už existujúceho systému Program manager, ktorý zahŕňa viacero systémov s rôznym zameraním pre podnikové účely. Od tohto zámeru sa neskôr ustúpilo a rozhodlo sa, že sa PIS QAM stane samostatným systémom.

Na vývoj systému sme sa rozhodli použiť platformu Java Enterprise Edition (EE). Java EE je súbor technológií a API zameraných na podnikové aplikácie. Môže byť považovaná za rozšírenie Java Standard Edition (SE) pre podporu vývoja distribuovaných, robustných, výkonných a dostupných aplikácií. Takto vyvíjaná podniková aplikácia umožňuje jednoduchú prístupnosť pre používateľa. PIS QAM bude nasadený na serveri, na ktorý majú prístup všetky počítače v podniku v rámci intranetu. Nie je tak potrebná jeho inštalácia na množstvo počítačov, ako by to bolo v prípade standalone aplikácie. Zároveň pri vytvorení novej verzie nie je nutné preinštalovanie na všetkých počítačoch, ale len nasedenie novej verzie na server. Používateľovi je do PIS QAM umožnený prístup prostredníctvom webového prehliadača z ktoréhokolvek počítača, či mobilného zariadenia pripojeného do intranetu. Vďaka takto riešenej prístupnosti môže používateľ s PIS QAM pracovať prípadne aj cez smartfón alebo tablet bez nutnosti toho, aby sme vyvíjali ďalšiu aplikáciu pre mobilné zariadenia. Zabezpečenie intranetu podporuje zároveň aj bezpečnosť PIS QAM, keďže nikto mimo podnikov spoločnosti nemá povolený prístup k serveru a teda ani k PIS QAM. V rámci samotnej spoločnosti bude samozrejme bezpečnosť riešená ešte prostredníctvom používateľského manažmentu – používateľ sa musí prihlásiť a bude mať pridelené určité práva. Nevýhodou pripojenia cez intranet je, že v prípade poruchy siete nebude možné s PIS QAM pracovať.

Všetky dáta týkajúce sa zákaziek bude PIS QAM získavať zo SAP ERP. V prvých verziách je potrebné, aby ich používateľ manuálne zadal, neskôr budú

automatizovane získavané pomocou funkčných modulov. Dáta o meraniach budú získavané z meracích nástrojov. V prvých verziách bude taktiež potrebné manuálne zadanie dát, neskôr bude tento proces zautomatizovaný. Všetky potrebné dáta bude vyvíjaný systém uchovávať vo vlastnej databáze.



Obrázok 7 Návrh systému

3.4 ARCHITEKTÚRA VYVÍJANÉHO PIS QAM

Dôležitou súčasťou každého softvéru je jeho architektúra, ktorá tvorí základný skelet celého softvéru. Existuje viacero typov architektúr, z ktorých je potrebné si vybrať tú správnu pre vyvíjaný softvér.

Nami vyvíjaný PIS QAM je implementovaný ako webová aplikácia s využitím platformy Java EE, ktorá je vhodná práve pre vývoj rozsiahlych podnikových aplikácií.

Pre PIS QAM sme zvolili vrstvenú architektúru. PIS QAM je tak rozdelený na tri základné vrstvy a to *tl.qam.ui*, *tl.qam.business* a *tl.qam.persistence*, kde *tl.qam.ui* zabezpečuje používateľské rozhranie, *tl.qam.business* obchodnú logiku a *tl.qam.persistence* perzistenciu údajov. Vo vrstvenej architektúre sú funkcie systému usporiadané do samostatných vrstiev a každá vrstva sa spolieha iba na funkcie a služby poskytované vrstvou, ktorá leží pod ňou. Tento prístup umožňuje inkrementálny vývoj systému. Za predpokladu, že sa nezmení rozhranie, je možné vrstvu nahradiť inou

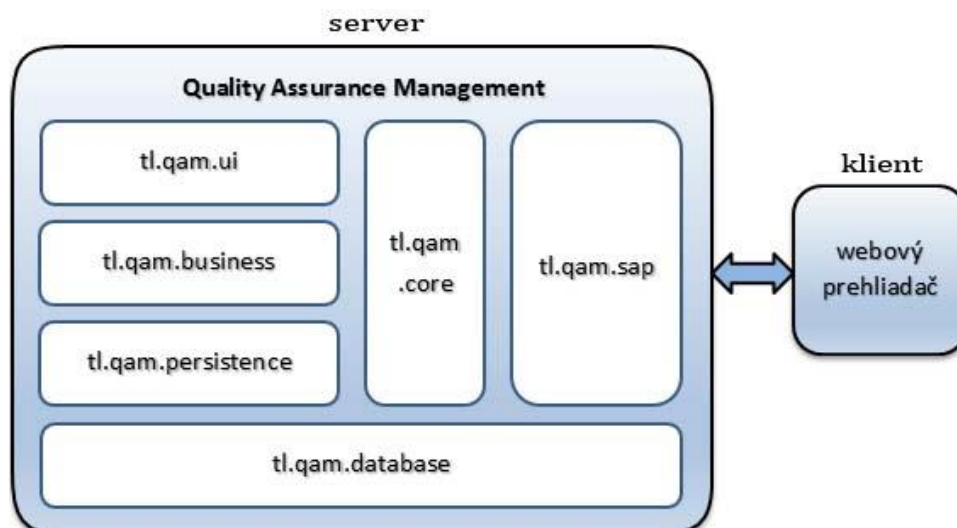
ekvivalentnou vrstvou. Naviac platí, že ak sa zmení rozhranie vrstvy alebo je vrstva rozšírená o nové funkcie, ovplyvní to iba susednú vrstvu (Sommerville, 2011).

Vzhľadom na rozsiahlosť PIS QAM a teda aj samotných vrstiev, bola každá vrstva implementovaná ako samostatný modul. Okrem spomínaných troch modulov sme vytvorili aj ďalšie. Modul *tl.qam.core* nám poskytuje všeobecné funkcie využívané pozdĺž všetkých vrstiev. V tomto module je napríklad implementované logovanie. Samostatný modul sme vytvorili aj pre zabezpečenie komunikácie so SAP ERP. Tento modul dostal názov *tl.qam.sap*. Za účelom uchovávania aktuálnej verzie štruktúry databázy v podobe skriptu na jednom mieste spolu s PIS QAM sme vytvorili aj modul *tl.qam.database*. *Tl.qam.database* neposkytuje žiadne funkcie a teda nie je potrebné, aby bol priamou súčasťou PIS QAM. Slúži len na uchovávanie a verzionovanie skriptu databázy spolu s PIS QAM.

Rozdelenie celého PIS QAM do vrstiev a modulov predstavuje spôsob, ako dosiahnuť oddelenie a nezávislosť jednotlivých častí celého systému. Pomocou takýchto častí si dokážeme vyskladať celý systém do výslednej želanej podoby. Moduly je možné skladať v horizontálnom aj vertikálnom smere. V horizontálnom smere moduly zabezpečujú rôzne úlohy. Vo vertikálnom smere zas moduly tvoria spomínané vrstvy systému. Takto zvolená architektúra má viacero výhod. Umožňuje oddelený vývoj jednotlivých modulov a zvyšuje sa aj prehľadnosť architektúry systému (rozdelenie zodpovednosti, komunikácia prostredníctvom rozhrania, riešenie komplexnosti) (Rábová, Drlík, 2013).

Pre zjednotenie týchto modulov do jedného celistvého systému sme využili nástroj Apache Maven. Apache Maven je nástroj pre riadenie a buildovanie projektov založených na programovacím jazyku Java. Zjednodušuje a štandardizuje celý proces buildovania, pričom umožňuje zbuildovať aj viacero projektov dokopy. Štruktúra Maven projektov je deklarovaná v súboroch Project Object Model (POM) typu xml. V *pom.xml* môžeme nastaviť pluginy používané nástrojom Apache Maven, závislosti projektu a v rámci konfigurácie buildovania aj typ balenia projektu, tzv. *packaging*. Nami vytvorené moduly majú podľa ich účelu nastavený rôzny typ balenia (*war*, *jar*, *ejb*), pričom ale vo výsledku sú všetky tieto balíky zabalené do balíka typu *ear*, ktorý nasadíme na server (apache.org, 2016).

Prístup k PIS QAM je následne realizovaný na princípe architektúry klient – server. Používateľ pristupuje k PIS QAM cez klienta, ktorým je v tomto prípade webový prehliadač a využíva funkcie na serveri nasadeného PIS QAM.



Obrázok 8 Architektúra PIS QAM

3.5 NÁVRH A VYTVORENIE DATABÁZY

Aby používateľ mohol pri práci s PIS QAM opätovne používať jednotlivé dáta, musia byť tieto uchovávané v databáze PIS QAM. Za databázový systém sme si zvolili Microsoft SQL Server 2012 s prostredím pre správu databázového systému MS SQL Server 2012 Management Studio.

Na vytvorenie návrhu databázy sme použili Enterprise Architect 10, pomocou ktorého sme vytvorili ERD diagram reprezentujúci databázu PIS QAM. Pri návrhu sme vychádzali aj z informácií, ktoré sme získali z rozhovorov. Na ich základe sme museli správne rozpoznať entity vyskytujúce sa v procesoch a aj ich atribúty, aby sme mohli o nich uchovávať dáta. Niektoré entity (napr.: zákazka, produkt, pobočka) existovali už aj v SAP ERP. S týmito sme museli náš návrh zjednotiť z toho dôvodu, že neskôr pri automatizovanom získavaní dát ich budeme na ne mapovať. Pri návrhu sme rozpoznali, že niektoré tabuľky budú len tzv. číselníky (enumy), a teda ich obsah bude preddefinovaný a nebude sa meniť vplyvom používateľovho používania PIS QAM. Týka sa to tabuľky Factory, ktorá uchováva dáta o pobočkách (podnikoch) vybranej spoločnosti a tabuľky ProductionOrderType reprezentujúcu typ zákazky. V týchto prípadoch sme museli zistiť aj konkrétne dáta, aby sme ich mohli do tabuliek vložiť.

Na základe databázového návrhu sme mohli vytvoriť fyzickú databázu. Počas nasledujúceho vývoja PIS QAM bola databáza ešte viackrát upravovaná z dôvodu vývojom pribúdajúcich nových funkcií PIS QAM, ale aj kvôli zmenám požiadaviek na PIS QAM. Posledná verzia databázy, s ktorou pracuje PIS QAM má 25 tabuliek, z toho je 8 cross tabuliek. Výsledný dátový model je zobrazený v prílohe B.

3.6 NÁVRH TRIED

Po tom, ako sme si stanovili požiadavky, namodelovali prípady použitia a identifikovali objekty vyskytujúce sa v danej doménovej oblasti a aj ich atribúty, môžeme toto všetko využiť na vytvorenie diagramu tried. Navrhli sme triedy ProductionOrder, Customer, Drawing, Size, Product, Article, Protocol reprezentujúce hlavné objekty domény. Keďže informácie o inštanciách týchto tried si potrebujeme uchovávať v databáze, navrhli sme ku každej tejto triede aj DAO (data access object) triedu zabezpečujúcu komunikáciu s databázou. Každá zákazka sa počas jej životného cyklu môže nachádzať v rôznych stavoch. Navrhli sme teda enum reprezentujúci tieto stavy. Trieda ProductionOrder obsahuje atribút state tohto typu. Vedeli sme, že budeme potrebovať aj zaznamenávať akcie vykonané používateľom v PIS QAM. Za týmto účelom sme navrhli triedu LastAction, ktorá má atribút description popisujúci vykonanú akciu. Pri výrobkoch bude potrebné pracovať aj s ich nameranými hodnotami, preto sme navrhli samostatnú triedu ArticleMeasurement a trieda Article obsahuje zoznam inštancií tejto triedy. Rátali sme už aj s tým, že aplikácia bude komunikovať so SAP ERP, preto sme navrhli triedu SAPService, ktorá bude túto komunikáciu zabezpečovať a prostredníctvom nej budeme získavať potrebné dáta.

Takto vytvorený diagram tried sa stal podkladom pre implementáciu systému. Z dôvodu dokumentácie systému sme diagram tried udržiavali aj počas vývoja, aby bol aktuálny. Diagram tried sa v priebehu vývoja veľakrát zmenil a niekoľkonásobne rozšíril. Výsledný diagram tried je zobrazený v prílohe C.

3.7 JAVA PERSISTENCE API (JPA)

Aby sme mohli v rámci systému pracovať so zákazkami, ale aj s inými objektmi, ktoré sme si definovali pri návrhu databázy a aj návrhu tried, musíme vytvoriť triedy reprezentujúce tieto objekty domény. Stav (hodnoty atribútov) inštancií tried je uchovávaný len kým beží Java Virtual Machine (JVM). V prípade, že sa JVM zastaví, alebo garbage collector vyčistí pamäť, inštancie sa stratia a s nimi aj ich stav. Niektoré inštancie ale potrebujeme, aby boli perzistentné, to znamená že ich stav sa uchováva na pamäťovom médiu a tento stav môže byť znovu kedykoľvek použitý. Za účelom uchovávania dát využívame databázu.

Práca s dátami pri databáze ale nie je totožná s prácou, ktorá sa používa pri objektovo-orientovaných programovacích jazykoch akým je aj Java. V databáze sú entity uložené ako riadky tabuľky a vzťahy medzi entitami sú vyjadrené cudzími

klúčmi. Java takýto slovník nepozná. V Jave pracujeme s inštanciami tried, ktoré môžu dediť od ostatných, alebo môžu aj ukazovať na kolekcie iných inšancií. Princípom objektovo-relačného mapovania (object-relational mapping, ORM) je zosúladenie týchto dvoch technológií. Za týmto účelom sme použili Java Persistence API (JPA). Silnou stránkou JPA je, že nám umožňuje pracovať s entitami ako s bežnými objektami. JPA nám umožňuje ukladať entity do databázy, upravovať ich, vymazávať, ale aj sa na ne rôzne dotazovať pomocou Java Persistence Query Language (JPQL).

Pred samotným využívaním databázy akoukoľvek webovou aplikáciou je nevyhnutné pridať Java Database Connectivity (JDBC) driver do aplikačného servera. V našom prípade sme si zadovážili JDBC driver pre MS SQL Server. Ďalším dôležitým krokom je vytvorenie pripojenia z aplikačného serveru (v našom prípade AS WildFly) na databázu pridaním datasource-u.

3.8 JPA ENTITY

Triedy, ktorých inštancie ukladáme do databázy, sú označené anotáciou `@Entity`. Takáto trieda v Jave reprezentuje databázovú tabuľku. Aby mohla byť do databázy uložená inštancia triedy, musí byť táto trieda namapovaná na databázovú tabuľku, do ktorej budeme zapisovať alebo z nej čítať. Využívajú sa na to mapovacie anotácie. Anotáciou `@Table` a jej parametrom `name` špecifikujeme databázovú tabuľku, na ktorú je namapovaná JPA entita. Atribúty sa mapujú na stĺpce databázovej tabuľky pomocou anotácie `@Column` a jej parametra `name` predstavujúceho názov stĺpca. Atribúty musia byť `private` alebo `protected` a prístupuje sa k nim pomocou metód `get` a `set`. Tak ako aj v databáze, aj na atribúty JPA entity môžeme nastaviť rôzne obmedzenia ako napríklad pri použití anotácii `@NotNull` a `@Size`. Anotáciou `@Basic` s parametrom `optional` môžeme taktiež nastaviť, či daný atribút môže byť `null`. Každá JPA entita musí mať jedinečný identifikátor s anotáciou `@Id` reprezentujúci primárny kľúč. Anotácia `@GeneratedValue` na atribúte identifikátora zabezpečuje generovanie jeho hodnoty. JPA entita musí tiež mať prázdny `public` alebo `protected` konštruktor a môže mať aj iné konšuktory. Trieda musí implementovať rozhranie `Serializable` a mať atribút `serialVersionUID`. Metódy `hashCode()` a `equals(Object other)` musia byť taktiež implementované v danej triede.

Na JPA entite sú definované aj `NamedQueries`, čo sú pomenované JPQL dotazy, ktorými sa dotazujeme na danú entitu v databáze.

V nasledujúcich riadkoch je zobrazená ukážka kódu vyššie popisovanej JPA entity. V ukážke je zobrazená iba jedna NamedQuery (v pôvodnom kóde je ich viac).

```
@Entity
@Table(name = "T_ProductionOrder")
@NamedQueries({
    @NamedQuery(name = "ProductionOrder.findOrdersWithState",
        query = "SELECT p FROM ProductionOrder p WHERE p.state IN :states")
})
public class ProductionOrder implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;

    @Basic(optional = true)
    @Size(max = 250)
    @Column(name = "description")
    private String description;

    @NotNull
    @Basic(optional = false)
    @Column(name = "state")
    @Enumerated(value = EnumType.STRING)
    private POStateEnum state;

    @ManyToOne(cascade = CascadeType.ALL, optional = true, fetch = FetchType.EAGER)
    @JoinColumn(name = "fk_customer_id", referencedColumnName = "id")
    private Customer customer;
```

JPA entita ProductionOrder, ktorú sme vytvorili pre reprezentáciu zákazky, má aj atribút state, ktorý je typu POStateEnum. POStateEnum je nami vytvorený typ enum definujúci stavy, ktoré môže zákazka nadobúdať. Databáza samozrejme tento enum nepozná, keďže si ho v databáze neuchováame ako tabuľku. V takomto prípade použijeme anotáciu @Enumerated, ktorá špecifikuje, že sa jedná o enum hodnotu a pomocou EnumType definujeme, ako má byť táto hodnota uchovávaná.

Ak vytvárame JPA entity, ktoré sú v databáze prepojené cudzím kľúčom, v Jave môžeme tento vzťah reprezentovať anotáciami @OneToOne, @OneToMany, @ManyToOne alebo @ManyToMany. Pri týchto anotáciách môžeme nastaviť aj spôsob kaskádovania (cascade) alebo načítavania (fetch). V prípade vzťahu zákazky a zákazníka je to tak, že zákazník môže mať viacero zákaziek, ale jedna zákazka môže patriť iba jednému zákazníkovi. Preto sme v JPA entite ProductionOrder pre atribút Customer použili anotáciu @ManyToOne. Kaskádovanie je nastavené na CascadeType.ALL, čo znamená, že akákoľvek predchádzajúca operácia vykonaná na danej entite zákazníka bude vykonaná aj na entite jeho zákazky. Čiže, ak z databázy zmažeme konkrétnu entitu zákazníka, vymažú sa aj všetky jeho zákazky.

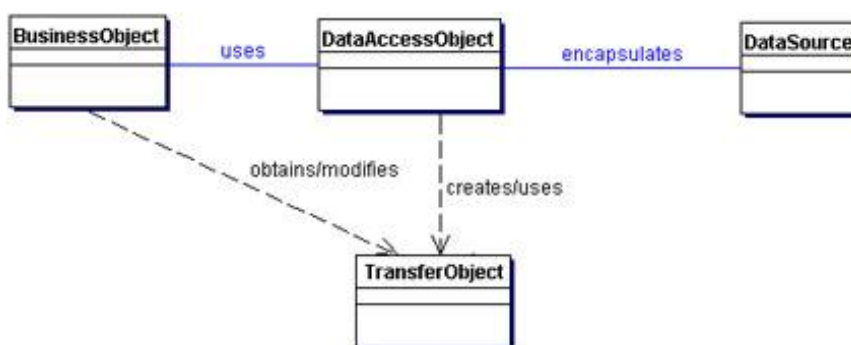
FetchType.EAGER znamená, že vždy, keď načítavame zákazku z databázy, načíta sa aj tento atribút (opakom je lazy načítavanie).

Vyššie popisovaným spôsobom sme vytvorili všetky potrebné JPA entity. Vytvorili sme aj enumy, ktoré sú využívané v rámci týchto JPA entít.

3.9 NÁVRHOVÝ VZOR DATA ACCESS OBJECT (DAO)

Prístup k dátam realizujeme s využitím návrhového vzoru Data Access Object (DAO). DAO slúži na zovšeobecnenie a zapuzdrenie všetkých prístupov k dátovým zdrojom. DAO zároveň riadi pripojenie na dátové zdroje za účelom získavania, ale aj ukladania dát. V našom prípade je tým dátovým zdrojom databáza. DAO vytvára abstraktnú vrstvu medzi biznis vrstvou a vrstvou perzistencie. Výhodou DAO je, že úplne skrýva implementačné detaily prístupu na dátový zdroj pred klientom (biznis vrstvou). DAO poskytuje klientovi rozhranie pre prácu s dátami na úrovni vrstvy perzistencie. Klient len volá určitú metódu, pričom nevidí, ako je táto implementovaná. To umožňuje zmeny a prispôsobenie sa dátovému zdroju bez toho, aby to malo vplyv na klienta. V podstate DAO pôsobí ako adaptér medzi klientom a zdrojom dát (oracle.com; Carman, 2002).

Primárnym objektom tohto návrhového vzoru je rovnomenný objekt DataAccessObject. Práve tento DAO objekt implementuje prístup k dátovým zdrojom a BusinessObject vykonáva dátové operácie, ako sú načítavanie a ukladanie dát, prostredníctvom tohto objektu (oracle.com).



Obrázok 9 Diagram tried reprezentujúci vzťahy v návrhovom vzore DAO⁷

3.10 PRÁCA S ENTITAMI

Priamym dátovým zdrojom pre PIS QAM je databáza. Prácu s entitami, ktoré sú v nej uložené, alebo ktoré chceme do nej ukladať, realizujeme cez DAO objekty. DAO

⁷ Zdroj Obrázok 9: <http://www.oracle.com/ocom/groups/public/@otn/documents/digitalasset/146804.jpg>

objekt implementuje spôsob, akým sa k dátam (entitám) pristupuje a ako sa s nimi pracuje. DAO objekty sme vytvorili pre každú našu JPA entitu.

S každou entitou by sme mali byť schopní vykonať základné CRUD operácie: vytvorenie entity (create), jej prečítanie (read), aktualizovanie (update) a vymazanie (delete). Z tohto dôvodu sme si vytvorili abstraktnú triedu `AbstractDAO.java`, z ktorej dedia všetky ostatné triedy DAO objektov pre konkrétne JPA entity.

3.10.1 EntityManager

Vykonanie akejkoľvek databázovej operácie nám zabezpečuje `EntityManager`. `EntityManager` je hlavnou časťou JPA. Riadi životný cyklus entít, ako aj dotazovanie sa na entity v rámci perzistentného kontextu. S využitím `EntityManager`a dokážeme vytvárať a odstraňovať entity z databázy, vyhľadávať entity podľa primárneho kľúča a taktiež umožňuje spúšťať rôzne databázové dotazy.

Aby sme mohli využiť metódy, ktoré sú poskytované `EntityManager`om, musíme vytvoriť v DAO objekte inštanciu `EntityManager`a s anotáciou `@PersistenceContext`. Každú inštanciu `EntityManager`a si môžeme predstaviť ako objekt triedy implementujúcej rozhranie `EntityManager`, ktorý nám umožňuje pracovať v určitom perzistentnom kontexte, s ktorým je asociovaný. Perzistentný kontext definuje oblasť, v rámci ktorého môžeme použiť inštanciu `EntityManager`a na pristupovanie a manipuláciu s entitami. Perzistentný kontext obsahuje súbor entít, ktoré sú spracovávané konkrétnou inštanciou `EntityManager`a. Vytváranie, upravovanie, vyhľadávanie podľa primárneho kľúča a vymazávanie entít sme následne realizovali volaním metód `EntityManager`a `persist`, `merge`, `find` a `remove`.

3.10.2 JPQL dotazy

Práca, ktorú sme potrebovali s entitami vykonávať, presahuje základné CRUD operácie. V databázovom svete sa práca s relačnými dátami realizuje pomocou SQL jazyka, ktorého syntax je tabuľkovo orientovaná. V Jave používame taký jazyk, ktorý je prispôsobený na prácu s objektami. Práve takým jazykom je Java Persistence Query Language (JPQL). JPQL je jazyk definovaný v JPA na dotazovanie sa entít uchovávaných v relačných databázach. Syntax JPQL je podobná syntaxi SQL, ale je prispôbena viac na prácu s objektami než databázovými tabuľkami. JPQL nepozná databázovú štruktúru, nepracuje s tabuľkami a stĺpcami, ale s objektami a atribútmi.

JPQL dotazy môžeme integrovať do kódu v jazyku Java cez tzv. queries. Existuje niekoľko rozdielnych typov queries definovaných v JPA. My sme používali

NamedQueries. NamedQueries patria medzi statické queries, ktoré narozdiel od dynamických queries môžu byť efektívnejšie na vykonávanie, pretože stačí ich preložiť iba raz a nie zakaždým, keď ich chceme vykonať. Na definovanie takýchto queries musíme použiť anotáciu @NamedQuery na JPA entite, ktorej sa dotaz týka. V rámci tejto anotácie vieme nastaviť názov pre NamedQuery a aj jej samotné znenie. Prostredníctvom názvu, ktorý sme si určili, budeme vedieť zavolať konkrétnu NamedQuery. Viacero queries typu NamedQuery môžeme na JPA entite definovať nasledovným spôsobom:

```
@NamedQueries({
    @NamedQuery(name = "ProductionOrder.findById",
        query = "SELECT p FROM ProductionOrder p WHERE p.id = :id"),
    @NamedQuery(name = "ProductionOrder.findBySapId",
        query = "SELECT p FROM ProductionOrder p WHERE p.sapId = :productionOrderSapId"),
    @NamedQuery(name = "ProductionOrder.findOrdersWithState",
        query = "SELECT p FROM ProductionOrder p WHERE p.state IN :states")
})
public class ProductionOrder implements Serializable {
```

Dotaz definovaný NamedQuery môžeme spustiť zavolaním metódy createNamedQuery() na vytvorenej inštancii EntityManagera. Ako parameter metódy zadáme názov konkrétnej NamedQuery, ktorú chceme spustiť.

Keď sme napríklad potrebovali v databáze vyhľadať konkrétnu zákazku na základe jej id, pod ktorým je uložená v SAP ERP, vytvorili sme si NamedQuery, ktorú sme pomenovali ProductionOrder.findById. Tento názov sme zadali do metódy createNamedQuery volanej z inštancie EntityManagera. Keďže sme potrebovali, aby nám na základe dotazu bol vrátený konkrétny objekt typu ProductionOrder, vytvorili sme si premennú query typu TypedQuery, v ktorej sme určili typ vráteného objektu. Ako môžeme vo vyššie zadanom kóde vidieť, v NamedQuery nie je priamo zadaná hodnota id, podľa ktorého budeme vyhľadávať, ale len parameter productionOrderSapId. Hodnotu tohto parametra nastavíme pomocou metódy setParameter volanej na query. Keďže id je jedinečný identifikátor, na základe ktorého vieme rozpoznať iba jeden objekt/entitu, aj nami vytvorená NamedQuery nám vráti ako výsledok len jeden objekt typu ProductionOrder. Tento výsledok získame cez príkaz query.getSingleResult();. V prípade, ak by náhodou v databáze nebola nájdená žiadna entita so zadaným id, odchyťavame pri danom príkaze výnimku NoResultException:

```

public ProductionOrder getProductionOrder(Long productionOrderSapId) {
    TypedQuery<ProductionOrder> query =
        em.createNamedQuery("ProductionOrder.findBySapId", ProductionOrder.class);
    query.setParameter("productionOrderSapId", productionOrderSapId);
    try {
        return query.getSingleResult();
    } catch (NoResultException ex) {
        return null;
    }
}

```

3.11 KOMUNIKÁCIA BACKENDU S FRONTENDOM

Na základe zvolenej architektúry PIS QAM používateľom vyvolané udalosti sú zachytávané na najvyššej vrstve a to vrstve používateľského rozhrania. Z tejto vrstvy sú následne volané metódy z biznis vrstvy a z biznis vrstvy je zas volaná nižšia vrstva – vrstva perzistencie.

3.11.1 Rozhranie biznis vrstvy

To, aké metódy budeme potrebovať volať v biznis vrstve z vrstvy používateľského rozhrania, sme si definovali vo vytvorenom rozhraní biznis vrstvy POInterface.java. Toto rozhranie nám umožňuje oddeliť volanie potrebných metód od samotnej implementácie týchto metód. Metódy rozhrania POInterface následne implementujeme v tzv. service triede ProductionOrderService.java. Vďaka takémuto riešeniu bolo možné na začiatku vývoja oddeliť vývoj frontendovej časti PIS QAM od jeho backendovej časti a obe časti vyvíjať paralelne. To bolo výhodou, lebo práve GUI bolo to, čo sme mohli hneď už aj na začiatku vývoja začať prezentovať a konzultovať s vedením oddelenia zabezpečovania kvality. Týmto spôsobom sa mohlo GUI vyvíjať v podstate nezávisle od backendovej časti PIS QAM, lebo nebolo potrebné čakať na dlhšie trvajúcu skutočnú implementáciu potrebných metód a celého backendu. Keďže ale bolo potrebné, aby sa v GUI zobrazovali dáta, no metódy, ktoré to mali zabezpečovať ešte neboli implementované, riešili sme to tak, že sme dočasne vytvorili ešte jednu service triedu, ktorá tiež implementovala rozhranie POInterface, no metódy boli v tomto prípade implementované povedzme len provizórne a obsahovali napevno zadané umelé dáta, ktoré boli zobrazované v GUI. Takto sme mali dve triedy implementujúce jedno rozhranie, čo by mohlo spôsobiť nedeterminizmus pri rozhodovaní, ktorá implementácia má byť vykonaná pri volaní metód rozhrania. Vyriešili sme to tak, že sme triedu ProductionOrderService, na ktorej prebiehala skutočná implementácia metód, dočasne označili anotáciou @Alternative, čím sme ju zablokovali. Neskôr, keď bola skutočná implementácia metód dokončená a v rámci

backendu bola integrovaná aj databáza, anotácia @Alternative bola odstránená a tiež sme odstránili aj dočasne vytvorenú service triedu. Tým bol úspešne ukončený zásadný krok na začiatku vývoja.

3.11.2 Návrhový vzor Data Transfer Object (DTO)

Pri realizácii prepojenia frontendovej časti PIS QAM s backendovou sme sa museli vysporiadať aj s prenosom dát medzi vrstvami. V niektorých prípadoch sme potrebovali na jednej obrazovke GUI zobrazit' viac dát než len tie, ktoré boli obsiahnuté v rámci jednej JPA entity. Bol to napríklad prípad, keď sme potrebovali na obrazovke GUI zobrazit' dáta konkrétneho vytvoreného protokolu a zároveň aj históriu vykonaných akcií s daným protokolom (sú to dve rozdielne JPA entity), ale aj iné dáta. Pre tieto prípady sme sa rozhodli použiť návrhový vzor Data Transfer Object (DTO). DTO nám zabezpečil to, že spomínaný prenos dát sme nemuseli realizovať viacerými volaniami metód naprieč všetkými vrstvami. Navyše celkový počet volaní by závisel aj od množstva volaných dát. Vytvorením DTO objektu sa o prenos dát naprieč vrstvami stará práve tento DTO objekt, ktorý zapúzdri dané dáta. Vo vyššie spomínanom prípade sme si vytvorili triedu ProtocolArchiveDTO, ktorá nám zapúzdрила všetky potrebné dáta, čiže okrem iného aj spomínané dve JPA entity. Žiadané dáta prechádzajú vrstvami v takomto tvare a následne sú z nich dáta získavané volaním metód get().

3.12 GRAFICKÉ POUŽÍVATEĽSKÉ ROZHRAŇIE

Práca s PIS QAM je umožnená cez grafické používateľské rozhranie (graphical user interface, GUI) bežiacie vo webovom prehliadači. Výhodou je, že webový prehliadač býva nainštalovaný na každom počítači. Nie je teda potrebné navyše robiť žiadne inštalácie, či konfigurácie na počítačoch, na ktorých bude PIS QAM používaný. Stačí len, aby používateľ zadal adresu PIS QAM nasadeného na serveri.

3.12.1 JSF, PrimeFaces, Ajax

Pri vývoji GUI bola využitá technológia JavaServer Faces (JSF). Technológia JSF bola vytvorená za účelom podpory tvorby grafických rozhraní. Aplikácie využívajúce JSF je možné si predstaviť ako štandardné webové aplikácie zachytávajúce HTTP požiadavky a následne vytvárajú HTML. V Java EE s využitím JSF môžeme vytvárať stránky využívajúce udalosti, listeneri a komponenty podobne ako v Java SE s technológiou Swing. Aplikácie využívajúce JSF majú používateľské rozhranie tvorené tzv. faceletmi. Facelety využívajú XHTML (Extensible Hypertext Markup Language),

ktorý je kombináciou HTML a XML. Facelety sú vytvorené z rôznych komponentov, ako sú napríklad textové polia, tlačidlá, tabuľky, atď. Webové aplikácie sú vytvorené z viacerých stránok – faceletov. Facelet je riadený špeciálnou Java triedou (tzv. backing bean), ktorá riadi prechod z jednej stránky na druhú, volá funkcie z nižších vrstiev a synchronizuje dáta s komponentami. JSF riadi pohyb dát z ale aj do GUI.

Spolu s JSF sme využívali aj PrimeFaces. PrimeFaces je open source sada komponentov s rôznymi rozšíreniami. PrimeFaces môžeme považovať ako rozšírenie pre JSF (Çivici, 2015).

Webové aplikácie poskytujú bohaté prostredie, ktoré môže byť náročné na dáta. Pri niektorých udalostiach je preto vhodné, aby nemusela byť aktualizovaná celá stránka, ale len jej konkrétna menšia časť (konkrétny komponent). Práve to nám zabezpečuje Ajax, ktorý je podporovaný v JSF a aj v PrimeFaces (Çivici, 2015).

3.12.2 Vývoj GUI pre PIS QAM

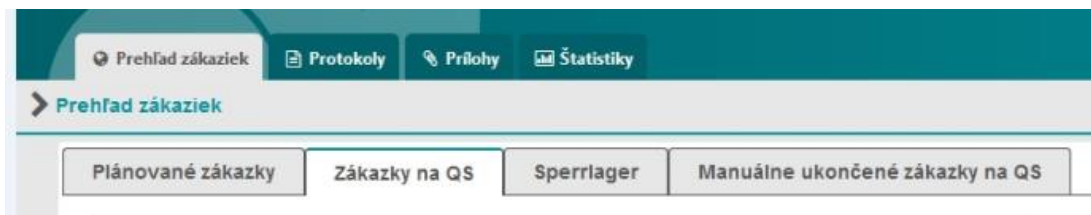
Od pôvodne navrhnutých a vytvorených faceletov pre vytvorenie novej zákazky, zobrazenie prehľadu aktuálnych a uzatvorených zákaziek, zobrazenie detailu zákazky, úpravu jej údajov a zobrazenie štatistík o zákazkách, prešlo celé GUI počas vývoja systému množstvom zmien. Vo výslednej verzii si používateľ môže pri práci so systémom na najvyššej úrovni zvoliť z pomedzi tabov Prehľad zákaziek, Protokoly, Prílohy a Štatistiky. Fotky GUI pre PIS QAM sú priložené v prílohe D. V nasledujúcich kapitolách bližšie popíšeme jednotlivé časti PIS QAM.

3.13 PREHLAD ZÁKAZIEK

Prvou a základnou funkcionalitou, ktorú má PIS QAM poskytovať, je sprehľadnenie a zefektívnenie práce so zákazkami. PIS QAM má umožňovať:

- manuálne vytvorenie zákazky v systéme,
- zobrazenie prehľadu všetkých zákaziek v systéme vo viacerých tabuľkách podľa aktuálneho stavu zákaziek,
- zobrazenie detailov konkrétnej zákazky,
- manuálne editovanie detailov zákazky.

Po vytvorení všetkých potrebných faceletov a aj ich backing bean, za ktorými sa nachádzajú ďalšie triedy z backendu, dostane používateľ po zvolení si hlavného tabu Prehľad zákaziek na výber ďalšie taby Plánované zákazky, Zákazky na QA, Sperrlager a Manuálne ukončené zákazky na QS.



Obrázok 10 Prehľad zákaziek

3.13.1 Prehľad zákaziek podľa stavu zákazky

Zákazka sa počas jej životného cyklu môže nachádzať v rôznych stavoch. Každý z vyššie spomenutých tabov prehľadne zobrazuje zákazky podľa ich stavov. V tabe Plánované zákazky sú v tabuľke zobrazené zákazky so stavom PLANNED – to znamená, že pri danej zákazke je plánované, že sa v budúcnosti dostane aj na oddelenie zabezpečovania kvality. V tabe Zákazky na QS sú zobrazené v tabuľke také zákazky, ktoré sa aktuálne nachádzajú na oddelení zabezpečovania kvality a teda sú v stave IN_QA alebo IN_QA_MAN podľa toho, či sa do tohto stavu dostali automaticky alebo manuálnym presunutím zákazky (stlačením tlačidla) používateľom z plánovaných zákaziek do zákaziek na QS. V tabe Sperrlager sú zákazky, ktoré sú na oddelení zabezpečovania kvality, ale boli pri nich zistené nejaké nedostatky. Ak kontrola kvality pre zákazku prebehla úspešne, používateľ môže danú zákazku ukončiť, stav zákazky sa zmení na QA_FIN_MAN a zákazka sa presunie do tabuľky v tabe Manuálne ukončené zákazky na QS. Je potrebné si uvedomiť, že neznamena to úplné uzavretie zákazky, ale len ukončenie jej jednej časti (jej spracovania oddelením zabezpečovania kvality) celého životného cyklu v podniku. Následne zákazka putuje na iné oddelenia podniku, prípadne sa môže aj znovu vrátiť na to isté oddelenie kvality alebo aj na oddelenie kvality inej dcérskej pobočky spoločnosti. Záleží to od konkrétnej zákazky. V týchto prípadoch zákazka nadobúda stavy OTHER_DEPT alebo OTHER_FACTORY. PIS QAM bude v spomenutých tabuľkách zobrazovať vždy iba tie zákazky, ktoré sú určené pre oddelenie kvality podniku, v ktorom je PIS QAM nasadený.

Stavy, ktoré môže zákazka nadobudnúť, reprezentuje enum POStateEnum. Následne v JPA entite ProductionOrder sa nachádza atribút state typu POStateEnum a podľa jeho hodnoty vieme zákazky rozdeliť podľa ich stavov do tabuliek. Hodnotu stavu určuje ďalší atribút, ktorý je v triede ProductionOrder a to downstreamOperations. Tento atribút obsahuje reťazec špecifických znakov, z ktorých každý predstavuje operáciu (resp. oddelenie, ktoré ju vykonáva), ktorá musí byť s danou zákazkou prevedená v priebehu jej spracovania. Operácie sú uvedené v poradí znakov zľava

doprava. Prvý znak vľavo predstavuje operáciu, ktorá nasleduje za práve vykonávanou operáciou. Keď príde zákazka na oddelenie, pracovník terminálom načíta čiarový kód z dokumentu Fertigungsauftrag, čím sa z atribútu downstreamOperations v SAP ERP odstráni prvý znak reprezentujúci dané oddelenie. Týmto spôsobom je určené ktorými všetkými oddeleniami má zákazka prejsť. Tak vieme určiť stav zákazky vzhľadom na oddelenie zabezpečovania kvality a zákazky rozdeliť do tabuliek v taboch.

3.13.2 Manuálne vytvorenie zákazky

Pre manuálne vytvorenie zákazky je pre používateľa k dispozícii GUI obrazovka vo forme formulára. Používateľ v tomto formulári vyplní všetky potrebné dáta týkajúce sa:

- zákazky vrátane termínov začatia, ukončenia a doručenia,
- produktu, ktorého sa zákazka týka,
- výkresu, ktorý popisuje produkt,
- zákazníka, ktorý zadal zákazku,
- zvolí si protokoly potrebné pre túto zákazku,
- zvolí si protokoly potrebné pre ďalšie zákazky týkajúce sa tohto produktu,
- pridať môže aj poznámky k produktu alebo k zákazke.

Používateľovi sme sa snažili zadávanie údajov zjednodušiť a zautomatizovať tým, že keď zadal číslo (jedinečný identifikátor) produktu, ktorý sa už v databáze nachádza, všetky polia týkajúce sa produktu sa automaticky predvyplnia podľa existujúcich dát. Zaznačia sa aj checkboxy protokolov, ktoré boli v niektorej z predošlých zakaziek označené ako požadované pre zákazky týkajúce sa tohto produktu. Automatické predvyplnenie platí aj pre dáta o zákazníkovi po zadaní čísla zákazníka a výkres po zadaní čísla výkresu a jeho verzie. V prípade výkresu sa skontroluje aj to, či už bola overená revízia tohto výkresu. Ak áno, checkbox určený pre tento atribút sa automaticky zaškrtnie a zároveň sa zablokuje jeho možnosť editovania, keďže, ak už raz bola overená revízia výkresu, nie je možné túto skutočnosť zmeniť.

Po stlačení tlačidla pre vytvorenie takto zadanej zákazky sa najskôr skontroluje, či sú vyplnené všetky polia, ktoré sú nevyhnutné pre vyplnenie. Ak nie, používateľ je na to upozornený zobrazením správy. Po vyplnení všetkých požadovaných polí môže byť zákazka úspešne vytvorená v PIS QAM a uložená v databáze. Z technického hľadiska nie je vytvorená a uložená v databáze len entita zákazky, ale aj produktu, zákazníka, výkresu a iných, ktoré súvisia s entitou zákazky. Bežnou situáciou je, že viacero

zákaziek má priradený ten istý výkres, produkt alebo zákazníka. Museli sme zabezpečiť, aby pri vytváraní takejto zákazky neboli znovu vytvorené aj už existujúce entity výkresu, produktu alebo zákazníka, a aby tak nedochádzalo k redundancii dát v databáze. Vytvorenie týchto entít je realizované tak, že najskôr sa skontroluje, či sa v databáze dané entity už nachádzajú. Ak áno, tak entity sa znovu nevytvoria, ale len sa do novo vytvorenej entity zákazky priradí cudzí kľúč na tieto ostatné už existujúce entity. Ak sa ale v databáze ešte nenachádzajú, tak bude vytvorená aj každá táto entita. O tom, či boli tieto objekty domény pridané do PIS QAM ako nové, alebo či len boli už existujúce priradené k novo vytvorenej zákazke, je používateľ taktiež informovaný zobrazením správy.

3.13.3 Detail zákazky a jeho editácia

Za účelom dodržania prehľadnosti v tabuľkách GUI pre zobrazenie prehľadu zákaziek, nemohli byť v týchto tabuľkách zobrazené všetky dáta týkajúce sa zákaziek, ale len tie najhlavnejšie. Všetky ostatné dáta sú používateľovi dostupné po otvorení detailu konkrétnej zákazky.

Okrem prezerania týchto dát, je používateľovi umožnená aj ich editácia. Nie pre všetky dáta je ale povolená ich zmena. Dáta, ktoré jednoznačne určujú zákazku, ako je číslo zákazky, produktu, výkresu, či zákazníka, určite nemôžu byť zmenené. S postupom vývoja sa zároveň aj redukoval počet dát, ktoré môžu byť upravené, až nakoniec bol počet zredukovaný na minimum a upravovať je možné iba počet vyrobených dielov, poznámky k produktu, popis zákazky a voľbu požadovaných protokolov pre túto zákazku, ale aj pre ďalšie zákazky týkajúce sa tohto produktu. Úplne zablokované je editovanie detailu pre zákazku, ktorá je už uzavretá. Takáto zákazka je už na oddelení zabezpečovania kvality celá vybavená, a preto by ani nemal byť žiadny dôvod pre jej editovanie.

Redukciou počtu atribútov, ktorým je možné meniť hodnoty, sme chceli tiež prispieť k odbremeneniu pracovníkov od ďalších povinností. Chceli sme zabezpečiť, aby bola zmena hodnôt atribútov, ktoré musia byť zmenené napríklad kvôli výskytu nejakej udalosti, vykonaná automatizovane. Týkalo sa to aj atribútu overenia revízie, ktorého hodnota sa následne mení na základe biznis logiky. Overenie revízie je dôležitou súčasťou pracovných procesov, preto sú pracovníci na túto časť ich procesov v GUI PIS QAM upozornení osobitným grafickým prvkom. Ikona reprezentujúca svietiacu diódu mení farbu v závislosti od hodnoty príslušného atribútu zákazky. Ak bol

v systéme vytvorený nový výkres, tak hodnota jeho atribútu overenia revízie je null, čo znamená, že overenie revízie tohto výkresu nie je potrebné a „dióda“ teda nesvieti (má šedú farbu). Ak ale používateľ zaznačí, že pre zákazku je potrebné urobiť aj protokol o vykonaní kontroly prvej vzorky (first article inspection – FAI) a táto zákazka je prepojená s výkresom, pri ktorom ešte nebolo vykonané overenie revízie, atribút nadobúda hodnotu false a „dióda“ svieti na červeno, čím upozorňuje na tento fakt, ktorý je potrebné riešiť. Ak bolo vykonané overenie revízie, čo zistíme tým, že do PIS QAM používateľ nahral dokument, tzv. krycí list, hodnota atribútu sa zmení na true a „dióda“ svieti na zeleno. Toto je dôležitá informácia, lebo kým pracovníci oddelenia zabezpečovania kvality nemajú krycí list, nemôžu uzavrieť zákazku, a teda nemôže byť zatiaľ ani doručená zákazníkovi. Pred vytvorením PIS QAM, si pracovníci oddelenia zabezpečovania kvality overovali tento fakt tým spôsobom, že museli manuálne prehľadávať súbory v systéme súborov a hľadať tento krycí list. Pritom mohlo dochádzať aj k chybám spôsobeným nepozornosťou pracovníka. Nami vytvorený spôsob im zjednodušuje a sprehľadňuje túto záležitosť a tým aj šetrí čas.

Okrem overenia revízie PIS QAM dokáže používateľa informovať aj o ďalších činnostiach, ktoré ešte musí zamestnanec v rámci svojho pracovného procesu spraviť, aby mohol danú zákazku úspešne ukončiť. Týka sa to najmä záležitostí ohľadne protokolov, preto sa tomu bližšie budeme venovať v kapitole určenej pre protokoly.

3.13.4 Zjednotenie formátu dátumov a času

Pri zákazkách, ale aj v iných častiach systému zobrazujeme dátumy, ktoré sa týkajú buď vytvorenia zákazky, dodania zákazky a iné. Aby sme zjednotili formát všetkých dátumov zobrazovaných v GUI, vytvorili sme si tzv. properties súbor datePatterns.properties, v ktorom sme ako kľúče zadali „time“, „date“, „datetime“ a hodnoty pre tieto kľúče sú konkrétne formáty v akom má byť čas alebo dátum zobrazený (napr. pre „date“ je to „dd-MM-yyyy“). V prípade potreby zadania formátu len zavoláme nasledujúcu metódu:

```
private String getDatePattern(String key) {
    Properties datePatternsProperties = new Properties();
    String datePattern = null;
    try {
        datePatternsProperties.load(this.getClass()
            .getResourceAsStream(Constants.DATE_PATTERNS_PROPERTIES_FILE));
        datePattern = datePatternsProperties.getProperty(key);
    } catch (Exception e) {
        mainLogger.error("Cannot open " + Constants.DATE_PATTERNS_PROPERTIES_FILE, e);
    }
    return datePattern;
}
```

, kde Constants predstavuje nami vytvorenú triedu, ktorá obsahuje cestu k properties súboru uloženú v triede ako konštantu DATE_PATTERNS_PROPERTIES_FILE.

Takto nemusíme napevno zadávať formát dátumu alebo času pri každom komponente, v ktorom je zobrazovaný, čím nielen zjednotíme formáty, ale aj šetríme čas pri prípadnej zmene formátu, kedy nám stačí prepísať formát iba na jednom mieste.

Pri dátumoch sme museli riešiť aj problém, keď sme si všimli, že dátum zobrazený v GUI a dátum uložený v databáze sa líšia. Problém sa nám podarilo vyriešiť zmenou JDBC driveru a určením timezone na komponentách zobrazujúcich dátumy.

3.14 ZAZNAMENÁVANIE VYKONANÝCH AKCIÍ

Jednou z funkcií, ktorú PIS QAM podporuje, je aj zaznamenávanie vykonaných akcií alebo udalostí, čiže tzv. logovanie. Logovanie je dôležité a užitočné nielen pre používateľa, ale tiež pre systémového administrátora. Podľa toho, pre koho je zaznamenaná informácia o vykonaní akcie určená, je aj rozdielne implementované jej zaznamenávanie.

Záznamy o rôznych vykonaných akciách, či vyskytnutých sa udalostiach, sa zaznamenávajú v súboroch, tzv. logoch. Systémový administrátor tak v prípade vyskytnutia sa problému alebo chyby môže overiť v týchto logoch, v čom je vlastne problém, čo ho spôsobilo, prípadne, čo mu predchádzalo. Za týmto účelom sme si vytvorili vlastný logger s anotáciou @MainLogger. Logger si môžeme deklarovať v triede, kde ho chceme použiť pomocou anotácií @Inject @MainLogger, ktoré sú priradené na deklaráciu atribútu Logger mainLogger. Nasledovným spôsobom dokážeme pomocou tohto logera zaznamenať konkrétnu udalosť:

```
mainLogger.error("Error by saving uploaded file: " + file.getFileName()  
+ " to path: " + path, e);
```

Zaznamenávanie vykonaných akcií pre používateľa je nielen implementované iným spôsobom, ale aj spĺňa iný význam než to bolo v predošlom prípade. Používateľ môže vďaka zaznamenávaniu vykonaných akcií vidieť v GUI na vybraných JPA entitách, aké akcie boli s nimi vykonané. Týka sa to najmä príloh pridaných používateľom a protokolov vygenerovaných PIS QAM na podnet používateľa. Používateľ tak môže vidieť, či je daný fyzický súbor protokolu alebo prílohy v pôvodnom stave, alebo či si ho niekto iný už stiahol, prezeral, či upravoval. Používateľ vidí nielen to, aká akcia bola so súborom vykonaná, ale aj to, kedy presne a kto (ktorý používateľ) danú akciu vykonal. PIS QAM zatiaľ nepodporuje používateľský manažment, používatelia sa neprihlasujú do systému, a preto údaj

o používateľovi momentálne nie je skutočný. Ako ale môžeme vidieť, PIS QAM je už teraz na to pripravený a ráta sa s tým, že neskôr bude implementovaná aj funkcia používateľského manažmentu. Následne jednoduchou úpravou sa k záznamu o vykonanej akcii priradí aj informácia o aktuálne prihlásenom používateľovi. Záznamy pre používateľa uchováваме v databáze. Z tohto dôvodu sme pre každú JPA entitu, pri ktorej chceme zaznamenávať vykonané akcie, vytvorili aj JPA entitu reprezentujúcu change log, napr. `ProductionOrderChangeLog.java` a DAO triedu, napr. `ProductionOrderChangeLogDAO.java`, pomocou ktorej sú JPA entity ukladané do databázy, alebo z nej získavané.

3.15 JAZYKOVÉ MUTÁCIE

PIS QAM je vyvíjaný pre medzinárodnú spoločnosť za účelom jeho používania najmä v štátoch Nemecko a Slovensko. Kvôli tomu PIS QAM podporuje tri jazykové mutácie a to slovenskú, nemeckú a anglickú. Preklady boli zabezpečené profesionálnymi prekladateľmi, ale aj vedením oddelenia zabezpečovania kvality, vďaka čomu sú v PIS QAM použité už aj zaužívané odborné termíny z tohto odboru.

Preklady textov uchováваме v properties súboroch. Takto máme všetky preklady na jednom mieste, vieme ich ľahko vyhľadať, upraviť, alebo doplniť a aj pridať novú jazykovú mutáciu. Pre každú jazykovú mutáciu máme samostatný properties súbor. Názov properties súborov je tvorený slovom „lang“, za ktorým nasleduje podtržník a skratka jazykovej mutácie, napr. „lang_sk“. Jeden zo súborov má názov tvorený iba slovom „lang“. Tento súbor bude použitý v prípade, ak by sa požadovaná jazyková mutácia nenašla. V našom prípade tento súbor obsahuje anglické preklady. Properties súbory obsahujú dvojicu hodnôt a to kľúč a hodnota, kde kľúč predstavuje názov, pod ktorým nájdeme text určený na zobrazenie v GUI, čiže hodnotu v danom jazyku. Kľúče sú vo všetkých properties súboroch rovnaké. Líšia sa len hodnoty priradené ku kľúčom.

V prístupe k prekladom nám pomáha `ResourceBundle`, ktorý je bežne používaný v súvislostiach s jazykovými špecifikami. Umožňuje nám vyhľadanie správneho prekladu z properties súborov na základe zadaného kľúča, ku ktorému je priradená daná textová hodnota vo zvolenom jazyku.

Predvolený jazyk pre PIS QAM je určený na základe jazyka štátu, v ktorom je PIS QAM nasadený. Voľba predvoleného jazyka závisí od properties súbor `application.properties`, ktorý používame ako základný konfiguračný súbor PIS QAM.

Pred nasadením PIS QAM na akúkoľvek pobočku vybranej spoločnosti práve v tomto súbore zadáme označenie danej pobočky. Podľa toho vie PIS QAM následne zistiť štát, v ktorom sa pobočka nachádza a vybrať správnu jazykovú mutáciu. Samozrejme je ale aj umožnené, aby si používateľ prostredníctvom GUI zvolil jemu vyhovujúcu jazykovú mutáciu.

3.16 PROTOKOLY

Dôležitou súčasťou procesov oddelenia zabezpečovania kvality sú protokoly. Sú to rôzne meracie protokoly alebo osvedčenia, ktorých je niekoľko typov. Pred zavedením PIS QAM ich museli pracovníci zakaždým manuálne vytvárať a vyplňať v nich všetky údaje, pričom každá zákazka si môže vyžadovať hneď aj niekoľko rôznych protokolov. Nehovoriac o tom, že takto vytvorené protokoly mohli byť rôzne pozakladané alebo uložené v počítači, čím sa v nich strácal prehľad.

PIS QAM dokáže vytvoriť všetky protokoly a zároveň v nich aj automaticky vyplní potrebné údaje týkajúce sa daného protokolu (informácie o spoločnosti, zákazníkovi, produkte, meraniach). Týmto sa proces tvorby protokolov nielen značne urýchlil, ale aj všetky protokoly sú uložené na jednom mieste – v PIS QAM, kde si ich vie pracovník ľahko nájsť a aj stiahnuť, či prípadne upraviť a upravený protokol naspäť nahrať do PIS QAM.

3.16.1 Prehľad požadovaných protokolov

Keďže sa v podnikoch vybranej spoločnosti pracuje s veľkým počtom zákaziek a každá zákazka si môže vyžadovať aj niekoľko protokolov, je dôležité mať aj v tomto prehľad a mať zavedený určitý poriadok. Používateľovi je umožnené zaznačiť na obrazovke GUI detailu zákazky prostredníctvom checkboxov, aké protokoly je potrebné urobiť pre konkrétnu zákazku. Tieto sú potom zvýraznené na obrazovke GUI určenej špeciálne pre protokoly, kde je umožnené aj ich vytvorenie. Nutnosť vytvorenia špecifických protokolov sa môže opakovať pre zákazky, ktoré sú zamerané na rovnaký produkt. Preto je používateľovi umožnené zaznačiť aj protokoly, ktoré sú vždy potrebné pre zákazku týkajúcu sa daného produktu a v každej ďalšej takejto zákazke budú tieto protokoly automaticky zaznačené ako potrebné. Niektoré protokoly môžu byť výslovne vyžadované konkrétnym zákazníkom pre všetky jeho zákazky. Táto skutočnosť je tiež viditeľná na tomto mieste prostredníctvom checkboxov, ktoré ale nie je možné editovať v GUI používateľom.

Protokoly			
	Požiadavka zákazníka	Požiadavka pre daný produkt	Požiadavka pre aktuálnu zákazku
Jazyk protokolu: DE			
Osvedčenia			
Osvedčenie o zhode (DIN 2.1)	☒	☐	☒
Osvedčenie o skúške (DIN 2.2)	☐	☐	☒
Akceptačný certifikát (DIN 3.1)	☐	☐	☐
Potvrdenie materiálu	☐	☐	☐
Protokoly			
Akceptačný certifikát (DIN 3.1) s meraniami	☒	☐	☒
MPS Protokol	☐	☐	☐
MPS-CPK Protokol	☐	☐	☐
MPS kontrolný plán	☐	☒	☒
Protokol o prvej vzorke	☐	☐	☐

Obrázok 11 Prehľad požadovaných protokolov

3.16.2 Implementácia funkcionality práce s protokolmi

Pre všetku spomínanú prácu, ktorú je možné s protokolmi vykonávať, je vyhradená samostatná obrazovka GUI (facelet protocols.xhtml). Na tejto jednej obrazovke má používateľ možnosť vytvoriť ktorýkoľvek protokol, pričom postačuje len jedno kliknutie na tlačidlo. Na obrazovke sú prehľadne zobrazené aj všetky protokoly vytvorené pre konkrétnu zákazku. Používateľ v tomto prehľade vidí názov protokolu, jazyk, v ktorom je protokol vytvorený, čas, kedy bol protokol vytvorený a autora, ktorý protokol vytvoril. Po kliknutí na tlačidlo si môže používateľ prezrieť v dialógovom okne aj históriu vykonaných akcií s konkrétnym protokolom (vytvorenie, stiahnutie, nahranie,...), pričom tiež vidí aj dátum a čas vykonania akcie a aj jej autora. Na obrazovke protokolov sú zobrazené aj stručné informácie o zákazke, ktorej sa protokoly týkajú, zákazníkovi a produkte.

Vzhľad vytváraných protokolov ostáva pre každý typ protokolu rovnaký. Menia sa len v ňom vyplnené údaje. Na disk servera, na ktorom je PIS QAM nasadený, sme uložili šablóny všetkých protokolov. Šablóny určujú vzhľad a rozmiestnenie údajov v dokumente, ale samotné údaje v nich nie sú vyplnené. Informácie o týchto šablónach sú zároveň uložené ako entity v databázovej tabuľke T_ProtocolTemplate. Pri vytvorení konkrétneho protokolu sa na disku servera vytvorí kópia zodpovedajúcej šablóny a do tohto dokumentu sa vyplnia aj konkrétne údaje o spoločnosti, zákazníkovi, produkte

a vykonaných meraniach. Informácie o vytvorených protokoloch sa ukladajú do databázovej tabuľky T_ProtocolArchive.

Na vyplňanie údajov do protokolov sme využili rozhranie pre programovanie aplikácií (API) Apache POI. Do PIS QAM sme toto API integrovali ako závislosť definovanú v pom.xml nástroja Apache Maven. Apache POI poskytuje rozhranie pre prácu s dokumentmi protokolov. S využitím tohto API dokáže PIS QAM upravovať a zapisovať do protokolov automatizovane bez zásahu používateľa. Vyplňané dáta o spoločnosti, zákazníkovi a produkte sú získané z databázy PIS QAM. Pri implementácii funkcionality vytvárania protokolov sme rátali s tým, že časom môžu pribudnúť nové typy protokolov a preto sme sa snažili vytvoriť čo najuniverzálnejšie riešenie. Vytvorili sme si triedu `TemplateTransformator.java`, v ktorej sú implementované základné metódy využívané pri vyplňaní všetkých protokolov. Následne sme vytvorili triedy, ktoré sa špecializujú na vytváranie konkrétnych typov protokolov a ktoré zároveň dedia z triedy `TemplateTransformator.java`. V týchto potomkoch sú implementované metódy pre vyplnenie dát a formátovanie dokumentu konkrétného typu protokolu. Aby PIS QAM presne vedel, na ktoré miesta v dokumente má vypísať ktoré dáta, sme použili properties súbory. V nich máme namapované, ktorý údaj má byť zapísaný na ktoré miesto protokolu.

V protokoloch sú obsiahnuté aj informácie o tzv. article sizes, čiže aké merania majú byť uskutočnené pre daný produkt a aké tolerancie sú pri nich povolené. Zdrojom dát v tejto súvislosti je súbor z programu Yocton. Yocton súbor má tiež stanovenú štruktúru, ktorú poznáme a na základe ktorej vieme získať potrebné dáta zapísané na konkrétnych miestach v tomto súbore. Používateľ musí Yocton súbor do PIS QAM nahráť ako prílohu prostredníctvom obrazovky GUI špeciálne vytvorenej pre prácu s rôznymi prílohami. Následne PIS QAM dokáže z Yocton súboru získať dáta o meraniach konkrétného produktu, ktoré je potrebné vykonať a uloží ich do databázy. Získané dáta sú tiež automatizovane vypísané do protokolu. Takýmto automatizovaným vytváraním protokolov sa zvyšuje rýchlosť vykonania administratívnych úloh zamestnancov oddelenia kvality, o zavedení poriadku a systému do procesu nehovoriac.

Niektorí zákazníci majú špeciálne požiadavky nielen na výrobky, ale aj na protokoly. Pri zákazkách takéhoto zákazníka je umožnené vytvorenie okrem štandardných protokolov aj špeciálnych zákazníckych protokolov. Zákaznícky protokol nepredstavuje nový typ protokolu. Je to len graficky upravený štandardný protokol takým spôsobom, akým to vyhovuje konkrétnemu zákazníkovi. Pri vytváraní protokolu,

ktorý je takto preťažený zákazníkovým protokolom, je pracovníkovi umožnené zvoliť si v comboboxe (v Primefaces je komponent pomenovaný SelectOneMenu), či sa má vytvoriť štandardný protokol alebo zákaznícky protokol. Zákaznícky protokol konkrétneho zákazníka je samozrejme možné vytvoriť iba pre zákazku tohto zákazníka. Pracovníkovi pri takejto zákazke nie sú ponúknuté na vytvorenie zákazníckeho protokoly iného zákazníka. Pri vybavovaní zákazky zákazníka, ku ktorému v systéme nie sú priradené žiadne zákazníckeho protokoly, sa combobox nezobrazí a pracovník môže vytvoriť iba štandardné protokoly.

3.16.3 Kontrola procesov a informovanie pracovníka

Pracovníci oddelenia zabezpečovania kvality si musia dávať pozor aj na to, že ani jedna zákazka, ktorá sa týka špecifického produktu určeného špecifickým výkresom, nemôže byť odovzdaná na odoslanie zákazníkovi, kým oddelenie nedisponuje tzv. krycím listom k takémuto produktu. Používateľ musí byť na tento fakt upozornený. Podobne, ako pri overení revízie, aj v tomto prípade kvôli jasnej výstižnosti a zachovaniu jednotnosti dizajnu sme použili ikonu reprezentujúcu svietiacu diódu. Krycí list úzko súvisí s protokolom FAI. Dióda nesvieti, ak FAI protokol nie je potrebný, a teda ani krycí list. Červená farba diódy upozorňuje, že FAI je vytvorený, ale chýba ešte krycí list. Ak oddelenie disponuje aj krycím listom a bol nahratý do PIS QAM, dióda svieti na zeleno.

3.17 TEST VÝKONNOSTI A OPTIMALIZÁCIA

Pred pripojením PIS QAM na SAP ERP sme sa museli hlbšie zamerať na otázku výkonnosti PIS QAM. Pripojenie na SAP ERP môže spôsobiť značné zaťaženie PIS QAM z dôvodu spracovávaného veľkého množstva dát. Mohlo by to spôsobiť spomalenie PIS QAM, čo by bolo nežiadúce. Preto sme do prípravy pripojenia PIS QAM na SAP ERP zahrnuli testovanie výkonnosti.

Testovanie prebiehalo na počítači s procesorom Intel Pentium CPU G2030 3.00GHz, 4 GB RAM a 64-bitovým operačným systémom Windows 7 Professional.

3.17.1 Interceptor na meranie času

Na testovanie výkonnosti existujú nástroje na to určené. Tieto sú ale platené a vyžadujú si zakúpenie licencie. Preto sme sa rozhodli testovanie výkonnosti uskutočniť iným spôsobom a to vytvorením vlastného interceptora, pomocou ktorého

budeme merať čas vykonávania jednotlivých metód. Po zistení najdlhšie trvajúcich metód sa zameriame práve na ich optimalizáciu.

V Java EE sa interceptory používajú na umožnenie vyvolania metód interceptora v triede, ku ktorej je pridružený. Trieda interceptora obsahuje metódy, ktoré sú spustené v súvislosti s metódami iných tried, ku ktorým je daný interceptor priradený. V našom prípade to bude metóda, ktorá bude merať čas vykonávania iných metód (oracle.com).

Najskôr sme nasledovným spôsobom vytvorili tzv. interceptor binding anotáciu, t.j. anotáciu reprezentujúcu konkrétny interceptor:

```
@InterceptorBinding
@Inherited
@Target({ TYPE, METHOD })
@Retention(RUNTIME)
@Documented
public @interface TimePerformanceInterceptor {
}
```

Ako už z názvu vyplýva, interceptor binding anotácia nám pomáha spojiť inteceptor so želanými triedami alebo ich metódami. Takto vytvorenou anotáciou @TimePerformanceInterceptor sme označili triedu nami vytvoreného interceptora TimePerformanceMeasurementInterceptor.java. Zároveň sme na triedu interceptora pridali aj anotáciu @Interceptor. V triede interceptora TimePerformanceMeasurementInterceptor sme vytvorili metódu measureTime(), ktorá bude merať čas vykonávania inej metódy. Na túto metódu sme pridali anotáciu @AroundInvoke, čo znamená, že takto anotovaná metóda bude vyvolaná tesne pred vykonaním sledovanej metódy. Najskôr si zaznamenáme čas spustenia vykonávania metódy, názov metódy a názov triedy, v ktorej je testovaná metóda obsiahnutá. Následne je spustená sledovaná metóda. Po skončení vykonávania sledovanej metódy si znovu zaznačíme aktuálny čas. Na základe týchto dvoch zaznačených časov vypočítame čas trvania vykonávania sledovanej metódy. Výsledok vypíšeme do konzoly:

```
@AroundInvoke
public Object measureTime (InvocationContext ic) throws Exception {
    long startTime = System.currentTimeMillis();
    String target = "unknown";
    Object targetObject = ic.getTarget();
    if(targetObject!=null){
        target = targetObject.toString();
    }
    String targetMethod = ic.getMethod().getName();
    try {
        System.out.println("Class: "+target+" - BEFORE calling method :"+targetMethod );
        return ic.proceed();
    } finally {
        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;
        System.out.println("*** Elapsed time Interceptor: "+elapsedTime + " ***");
        System.out.println("Class: "+target+" - AFTER calling method :"+targetMethod );
    }
}
```

Po vytvorení interceptora môžeme pridať anotáciu `@TimePerformanceInterceptor` na celé triedy, alebo konkrétne metódy, ktoré chceme merať. Aby bol interceptor aktivovaný, musíme pridať tag `<interceptors>` do `beans.xml`, kde uvedieme náš interceptor. Po skončení testov môžeme interceptor jednoducho zablokovať zakomentovaním tohto tagu.

3.17.2 Generovanie entít v databáze

Aby sme mohli odsimulovať približne skutočné zaťaženie PIS QAM, ktoré nastane po pripojení na SAP ERP, a tak dostali aj relevantné výsledky z merania času vykonávania metód, museli sme databázu naplniť dostatočným množstvom dát.

Napísali sme skript na naplnenie databázy, ktorý nám vytvorí entity pre všetky tabuľky s náhodne vygenerovanými hodnotami stĺpcov. Hodnoty sú síce vygenerované náhodne, ale s ohľadom na dátový typ stĺpca a obmedzenia, ktoré sú stanovené na danom stĺpci, ako napríklad veľkosť, či rozsah. Keďže používame databázu MS SQL Server 2012, skript sme napísali v jazyku T-SQL, ktorý je podporovaný touto databázou. Výsledný skript má 308 riadkov, popíšeme preto v stručnosti iba jeho základné časti a princíp.

Pri písaní skriptu sme rátali s tým, že skript bude použitý viackrát, čiže nielen v prípade aktuálne vykonávaného testu, ale aj testov, ktoré sa budú znovu robiť neskôr v budúcnosti, na čo sme pri jeho tvorbe brali ohľad. Na začiatku skriptu sme si definovali konštanty a následne premenné používané v skripte. Konštanty slúžia ako konfigurácia skriptu, kedy nastavením ich hodnôt hneď na začiatku skriptu určíme pre všetky tabuľky koľko entít sa bude generovať. Brali sme do úvahy aj to, že v reálnej situácii určite nebude v každej tabuľke rovnaký počet entít, ale v niektorých tabuľkách bude ten počet aj niekoľkonásobne väčší. Napríklad nebude platiť, že jedna zákazka bude mať priradený iba jeden záznam o vykonaní nejakej akcii (change log), ale tých záznamov bude viac. Tento násobok je tiež možné nastaviť cez konštantu.

Väčšina hodnôt stĺpcov sú náhodne generované textové reťazce, čísla, alebo dátumy. Pri simulácii sme ale chceli byť dôsledný a chceli sme, aby sa vygenerované hodnoty čo do najväčšej miery podobali na reálne. Vedeli sme, že niektoré stĺpce tabuliek obsahujú vždy iba hodnoty (slová) z určitej množiny slov. Tieto hodnoty máme uložené iba v Java kóde ako enumy, preto sa do nich zo skriptu nevieme dostať. Z tohto dôvodu sme v skripte deklarovali dočasné tabuľky, do ktorých sme vložili spomínané dáta:

```

DECLARE @ChangeLogDescription TABLE (CL_description varchar(10))
INSERT INTO @ChangeLogDescription (CL_description)
VALUES ('created'),('modified'),('closed')

```

Aby sme po naplnení databázy dokázali simulovať reálnu prácu s PIS QAM, museli sme pri generovaní entít brať ohľad aj na vzťahy medzi entitami, a teda na pridelenie správnych cudzích kľúčov. Tieto už neboli generované ako náhodne generované čísla, ale boli náhodne vyberané z reálnych id už vygenerovaných entít:

```

SET @RandomFkCustomerId = (SELECT TOP 1 id FROM dbo.T_Customer ORDER BY NEWID())

```

Na základe vyššie spomínaných princípov sme napísali celý skript pre generovanie entít všetkých tabuliek. Ako ukážku môžeme uviesť časť skriptu pre generovanie entít zákazníkov:

```

-- *****
-- * GENERATE ENTITIES to T_Customer *
-- *****
-- value of customer_sap_id must be between 1 and 9999999999
SET IDENTITY_INSERT [dbo].[T_Customer] ON
SET @RowNumber = 0
SET @LowerLimit = 1
SET @UpperLimit = 9999999999
WHILE @RowNumber < @COUNT_OF_ENTITIES
BEGIN
SET @RowNumberAsString = CAST(@RowNumber AS VARCHAR(100))
SET @RandomBIG = ROUND(((@UpperLimit - @LowerLimit - 1) * RAND() + @LowerLimit), 0)
INSERT [dbo].[T_Customer] ([id], [customer_sap_id], [name], [city], [street], [country])
VALUES (@RowNumber, @RandomBIG, N'custName'+@RowNumberAsString,
N'custCity'+@RowNumberAsString, N'custStreet'+@RowNumberAsString,
N'custCountry'+@RowNumberAsString)
SET @RowNumber = @RowNumber + 1
END
SET IDENTITY_INSERT [dbo].[T_Customer] OFF

```

Pred spustením skriptu sme najskôr zadali, aby vygenerovalo 1000 zákaziek a v ostatných tabuľkách bolo vygenerovaných 1000 až 12000 entít, spolu to bolo 43000 entít. Toto generovanie trvalo 51 sekúnd. Neskôr sme zadali na vygenerovanie 10000 zákaziek a ostatných entít bolo od 10000 do 120000, kedy čas generovania dokopy 430000 entít bol už 47 minút.

Náhodné generovanie dát je príliš časovo náročné, preto sme sa rozhodli po vygenerovaní 430000 entít vytvoriť skript takejto databázy, v ktorej sú už obsiahnuté aj konkrétne dáta. Spustenie takéhoto skriptu, v ktorom nie je potrebné dáta náhodne generovať, je oveľa menej časovo náročné. Vykonanie tohto skriptu trvalo 5 minút.

3.17.3 Sledovanie zaťaženia databázy

Po naplnení databázy dátami sme okrem časovej dĺžky vykonávania jednotlivých metód v Java kóde sledovali aj zaťaženie databázy. Keďže dotazy pre

získavanie, vytváranie, úpravu, ale aj mazanie dát sú vykonávané priamo na databáze, je dôležité mať aj databázu správne optimalizovanú.

Pre sledovanie zaťaženia databázy sme použili nástroj SQL Server Profiler. Tento nástroj umožňuje nastaviť sledovanie, ktoré dáva pozor na aktivitu servera v určitých udalostiach, ako je vykonávanie príkazov jazyka T-SQL. SQL Server Profiler zachytáva udalosti, a tak umožňuje vidieť, aké príkazy bežia na SQL Serveri a aj jeho zaťaženie. Vďaka tomuto zachytávaniu udalostí, presne vieme, aký kód odosiela PIS QAM na server. Pomocou tohto nástroja sme vykonali aj trasovania, čo znamená, že sme vytvorili stopu týchto udalostí. Počas trasovania sme simulovali prácu s PIS QAM, pričom sme zrealizovali všetky jeho funkcie. Výsledkom trasovania je súbor formátu .trc, ktorý môžeme využiť v ďalšom nástroji a to Database Engine Tuning Advisor (Henderson, 2000).

3.17.4 Ladenie výkonu

Pri prvom testovaní výkonu sa PIS QAM a aj databáza nachádzali v stave, v akom sme ich vytvorili, pričom v databáze sme už mali vygenerovaných celkovo 43000 entít, z toho 1000 zákaziek. Pri prvom teste výkonnosti a aj ďalších testoch bola testovaná celá aplikácia a všetky jej funkcie. Po vykonaní prvého testu sa naše predpoklady naplnili. Najdlhšie trvajúce metódy boli práve tie, ktoré sa týkali získavania zákaziek z databázy. Vykonanie metódy `getActualProductionOrders()` trvalo približne 20 sekúnd a metódy `getClosedProductionOrders()` približne 5 sekúnd. Dlhšie vykonávanie metód samozrejme spôsobilo aj spomalenie načítania obrazoviek GUI prehľadu zákaziek.

Namerané časy sú neprípustné, a preto sme ich museli riešiť. Začali sme úpravou načítavania atribútov JPA entity `ProductionOrder`. `ProductionOrder` je prepojený s ďalšími JPA entitami, čo tiež zaťažuje načítavanie zákaziek. Takýmto atribútom, ktoré sú deklarované ako veľké objekty, alebo listy týchto objektov (napr. výrobkov, protokolov, produktov,...), sme zmenili `FetchTyp.EAGER` na `FetchType.LAZY`, čím sme určili, že budú načítavané tzv. lazy loadingom. Následne sme aj v triede `ProductionOrderService` vytvorili nové metódy na načítavanie zákaziek a s tým súvisiace `NamedQueries` na JPA entite. Zákazky už nebudú načítavané len podľa toho, či sú alebo nie sú v stave uzavretá, ale podľa konkrétneho stavu (`IN_QA`, `PLANNED`,...), ktorý bude aktuálne žiadaný pre zobrazenie. Tým sme zabránili

načítavaniu zákaziek, ktoré nie sú v danej chvíli nevyhnutne potrebné, a tak zredukovali množstvo načítavaných zákaziek.

Po takto vykonaných úpravách v kóde PIS QAM sme vykonali druhé testovanie výkonnosti systému. Tentoraz sme už v databáze mali celkovo 430000 entít, z toho 10000 zákaziek. Aj napriek 10 násobnému zväčšeniu obsahu databázy sa nám podarilo dosiahnuť približne 10 násobné lepšie výsledky, kedy načítanie aktuálnych zákaziek trvalo približne 2 sekundy a uzatvorených zákaziek približne 2 milisekundy.

Pre vyhodnotenie zaťaženia databázy sme použili nástroj Database Engine Tuning Advisor. Nástroj Database Engine Tuning Advisor umožňuje skúšať záťažové súbory, ako je súbor trasovania a pre dosiahnutie čo najlepšieho výkonu odporučiť zmeny vo fyzických štruktúrach databázy ako je pridanie, úprava, alebo vymazanie indexov, či indexovaných pohľadov. Tento nástroj tiež môže odporučiť, aké štatistické informácie je vhodné zhromažďovať pre podporu fyzických štruktúr. V našom prípade nám odporučil vytvorenie niekoľkých indexov a aj štatistík. Všetky tieto odporúčania zhrnie do jedného sql skriptu, ktorý nám už stačí len spustiť na našej databáze.

V prípade optimalizácie databázy sme opätovne zvážili aj stanovené rozsahy a veľkosti stĺpcov. Takto sa nám podarilo niektoré stĺpce zmenšiť napríklad z varchar(70) na varchar(15). Taktiež aj v prípade číselných dátových typov sme sa snažili použiť čo najmenší možný číselný dátový typ a našli sme prípady, kedy sme mohli pretypovať pôvodný int na smallint alebo tinyint.

Po týchto úpravách v databáze sme vykonali tretí test výkonnosti. Namerané časové hodnoty vykonávania metód sa nám podarilo ešte znížiť, aj keď už nie až tak výrazne ako to bolo v prvom prípade. Čas načítavania aktuálnych zákaziek sme znížili z približne 2000 ms na približne 1700 ms a čas načítavania uzatvorených zákaziek sme znížili z približne 200 ms na približne 100 ms.

3.18 PREPOJENIE SO SAP ERP

Počet zákaziek, s ktorými sa v systéme bude pracovať, sa pohybuje v tisíckach. Aby so zákazkou bolo možné v rámci PIS QAM pracovať, je nutné ju v PIS QAM vytvoriť. Avšak pri takomto množstve zákaziek je manuálne vytváranie zákaziek náročné, najmä časovo z dlhodobého hľadiska aj napriek automatizácii pri vyplňaní potrebných údajov počas vytvárania zákazky. Keďže všetky zákazky vybranej spoločnosti sú evidované v SAP ERP, rozhodli sme sa naň pripojiť PIS QAM za účelom automatického získavania zákaziek. Používateľ tak nebude musieť manuálne vytvárať

zákazky v PIS QAM. Zákazky v PIS QAM budú na základe dát zo SAP ERP automaticky vytvorené alebo aktualizované podľa toho, či už v ňom daná zákazka existuje alebo nie.

3.18.1 SAP Java Connector (SAP JCo)

Pre vytvorenie pripojenia z PIS QAM na SAP ERP sme použili SAP Java Connector (SAP JCo). SAP JCo je middleware komponent, ktorý vývojárom umožňuje vyvíjať komponenty kompatibilné s ABAP (Advanced Business Application Programming) alebo aplikácie v Jave. SAP JCo podporuje komunikáciu s aplikačným serverom (AS) ABAP v oboch smeroch: prichádzajúci smer (Java volá ABAP), odchádzajúci smer (ABAP volá Javu) (SAP, 2014).

Aby sme mohli SAP JCo integrovať v PIS QAM, bolo potrebné si uložiť súbor sapjco3.dll na pevný disk a nastaviť naň cestu. Konfigurácia pripojenia sa nastavuje pomocou tzv. destinácie. Na AS WildFly sme vytvorili súbor ABAP_AS_WITHOUT_POOL.jcoDestination. V ňom sme nastavením parametrov určili meno používateľa, cez ktorého sa budeme pripájať na SAP ERP, heslo, ale aj názov aplikačného serveru, na ktorý sa chceme pripojiť a iné.

3.18.2 Implementácia pripojenia na SAP ERP v Jave

Jeden z modulov v PIS QAM je modul pre zabezpečenie komunikácie so SAP ERP. Do tohto modulu budeme z iných častí PIS QAM pristupovať cez jeho rozhranie, ktoré sme vytvorili. V rozhraní SapInterface sme definovali všetky metódy, ktoré budeme potrebovať volať v rámci PIS QAM, napr. metóda pre získanie všetkých zákaziek konkrétneho podniku vybranej spoločnosti, alebo získanie iba konkrétnych požadovaných zákaziek, či iných dát.

Následne sme vytvorili triedu SapContainerService, ktorá implementuje toto rozhranie aj so všetkými jej metódami. V nej sme deklarovali konštantu DESTINATION_NAME, ktorá obsahuje názov súboru jcoDestination. Deklarovali sme aj pre pripojenie nevyhnutné atribúty destination a repository, ktoré sú typu JCoDestination a JCoRepository. Destinácie sú riadené a poskytované JCoDestinationManagerom. V konštruktoze triedy priradíme atribútom hodnoty nasledovným spôsobom:

```
public SapContainerService() throws JCoException {  
    destination = JCoDestinationManager.getDestination(DESTINATION_NAME);  
    repository = destination.getRepository();  
}
```

3.18.3 Funkčné moduly

JCoRepository obsahuje runtime metadáta pre funkčné moduly. Funkčný modul je vlastne funkcia, ktorú je potrebné v SAP ERP naprogramovať, aby mohla byť následne volaná a využívaná z Java aplikácie. Museli sme dôkladne premyslieť, aké dáta potrebujeme zo SAP ERP a pomocou koľkých funkčných modulov ich budeme získavať. Museli sme brať do úvahy aj to, že získavanie veľkého množstva dát, ako je napríklad aj získanie všetkých zákaziek, je výpočtovo náročná operácia, ktorá môže zaťažiť SAP ERP. Príliš veľké zaťaženie SAP ERP je nežiadúce, keďže SAP ERP je bežne používaný v podniku a nemôžeme teda akokoľvek obmedziť jeho prevádzku. Preto sme v niektorých prípadoch, kedy potrebujeme funkčný modul volať každých niekoľko minút, obmedzili množstvo dát, ktoré žiadame zo SAP ERP. Obmedzením myslíme to, že napríklad pri funkčných moduloch, ktoré zisťujú, ktoré všetky zákazky sú plánované alebo aktuálne pre daný podnik vybranej spoločnosti, nevyžadujeme zo SAP ERP všetky dáta o zákazkách, ale len ich SAP id, podľa ktorého dokážeme zákazku v PIS QAM vyhľadať a aktualizovať jej stav.

Každý funkčný modul má svoj názov, pomocou ktorého ho budeme volať. Z tohto dôvodu sme si vytvorili enum SapFunctionModules, v ktorom sme definovali všetky funkčné moduly, ktoré budeme využívať. Vytvorili sme v ňom aj konštantu moduleName. Enum tak obsahuje nami definované názvy funkčných modulov, ako napríklad GET_ALL_PRODUCTION_ORDERS, z ktorých v každom je obsiahnutý aj názov funkčného modulu, pod ktorým je poznaný v SAP ERP, ako napríklad ZRFC_ZPP_TRACE. Urobili sme tak aj z toho dôvodu, že pri čítaní Java kódu, bude pre ktoréhokoľvek Java programátora kód zrozumiteľnejší, keď v ňom bude použitý nami zvolený názov, než keby sme použili originálny názov funkčného modulu.

Funkčný modul môže mať definovaný jeden alebo viacero vstupných parametrov. Funkčný modul GET_ALL_PRODUCTION_ORDERS má ako vstupný parameter JCo tabuľku podnikov, pre ktoré chceme získať dáta o zákazkách. Metóda getAllSapProductionOrders(), v rámci ktorej voláme spomínaný funkčný modul, obsahuje preto tiež parameter a to List, v ktorom sú obsiahnuté objekty typu String označujúce konkrétne podniky. V metóde sme deklarovali konštantu function, ktorá predstavuje funkčný modul, ktorý budeme volať:

```
final JCoFunction function = getFunction(SapFunctionModules.GET_ALL_PRODUCTION_ORDERS);
```

Následne sme vytvorili JCo tabuľku podnikov, ktorá reprezentuje vstupný parameter pre funkčný modul. V cykle sme prešli všetky podniky, ktoré sme dostali z parametra metódy a pridali sme ich do tejto JCo tabuľky:

```
JCoTable plantTable = function.getImportParameterList().getTable("I_WERKS");
```

Tabuľku sme určili ako vstupný parameter pre funkčný modul a spustili sme príkaz pre vykonanie funkčného modulu:

```
function.getImportParameterList().setValue("I_WERKS", plantTable);  
function.execute(destination);
```

Výstup z funkčného modulu dostaneme v podobe výstupného parametru, ktorý si uložíme do novej JCo tabuľky sapAllProductionOrdersJCoTable:

```
final JCoTable sapAllProductionOrdersJCoTable =  
    function.getExportParameterList().getTable("EX_ZPP_CAUFV");
```

Zámernne sme aj do názvu JCo tabuľky pridali „JCoTable“, aby sme vedeli jasne rozlíšiť túto JCo tabuľku, ktorú sme získali ako priamy výstup, od nami vytvorenej „tabuľky“ sapAllProductionOrdersTable. Metóda getAllSapProductionOrders() vracia SapContainer so všetkými zákazkami získanými zo SAP ERP. SapContainer je nami vytvorená generická trieda, ktorú využívame ako všeobecný kontajner pre akýkoľvek objekt získaný zo SAP ERP:

```
return new SapContainer<SapAllProductionOrdersTable>(sapAllProductionOrdersTable);
```

3.18.4 Konvertovanie z JCoTable

Pred tým než vrátime SapContainer so získanými dátami z metódy, v rámci ktorej voláme funkčný modul, je potrebné prekonvertovať JCo tabuľku a v nej obsiahnuté dáta na nami vytvorenú „tabuľku“, ktorá bude obsahovať objekty dátových typov bežne používaných v Jave. Je to z toho dôvodu, aby sme mohli komfortne pracovať so získanými dátami ako s bežnými objektmi v Jave.

Podobne ako pri JPA entitách, aj v tomto prípade sme si museli vytvoriť triedy, ktoré budú predstavovať prekonvertované JCo tabuľky a v nich obsiahnuté dáta. Vytvorili sme tak triedu SapAllProductionOrdersData, ktorá obsahuje zapuzdrené všetky atribúty, ktoré sú obsiahnuté aj v získanej JCo tabuľke, ale s tým rozdielom, že dátové typy sme nastavili také, ktoré v Jave korešpondujú s tými, ktoré boli dátam nastavené v SAP ERP, presnejšie v ABAP kóde. Okrem atribútov je v triede vytvorený konštruktor a aj getter a setter pre každý atribút.

Keďže JCo tabuľka sapAllProductionOrdersJCoTable obsahuje viacero zákaziek, nestačí nám len trieda reprezentujúca jednu zákazku získanú zo SAP ERP.

Potrebuje si vytvoriť aj ďalšiu triedu SapAllProductionOrdersTable, ktorá obsahuje zoznam týchto zákaziek v podobe listu objektov typu SapAllProductionOrdersData. Až táto trieda v našom prípade korešponduje s JCoTable a môžeme danú JCo tabuľku nahradiť takto nami vytvorenou „tabuľkou“.

Aby sme tak ale mohli urobiť, musíme prekonvertovať všetky dáta z JCo tabuľky do nami vytvorenej „tabuľky“. Metódy pre konvertovanie dát z JCo tabuľky do našich „tabuliek“ sme implementovali v triede SapUtils. Ako príklad uvedieme metódu JCoTable2SapAllProductionOrders so vstupným parametrom JCo tabuľkou, ktorú chceme konvertovať. Metóda obsahuje cyklus v rámci ktorého prejdeme každý riadok JCo tabuľky. Pre každý riadok vytvoríme nový objekt zákazky a nastavíme jej všetky atribúty podľa hodnôt získaných z JCo tabuľky. Pre správne priradenie hodnôt je nutné poznať aj názov, pod ktorým je daný atribút uložený v JCo tabuľke. Modifikáciu nasledovného príkazu zopakujeme pre každý atribút objektu zákazky:

```
sapAllProductionOrdersData.setOrderNumber(jcoTable.getString("AUFNR"));
```

Na konci každej iterácie cyklu pridáme prekonvertovanú zákazku do listu v objekte triedy SapAllProductionOrdersTable. Po skončení cyklu máme prekonvertovanú celú tabuľku, ktorú môžeme z metódy vrátiť ako výsledok.

Následne môžeme toto konvertovanie vyvolať priamo z metódy, v ktorej spúšťame funkčný modul, pred tým než z nej vrátime SapContainer s prekonvertovanou tabuľkou:

```
SapAllProductionOrdersTable sapAllProductionOrdersTable =  
    SapUtils.JcoTable2SapAllProductionOrders(sapAllProductionOrdersJCoTable);
```

Postup sme dodržiavali pri implementovaní každého funkčného modulu.

3.18.5 Plánovanie úloh na pozadí

Po implementovaní pripojenia na SAP ERP a aj funkčných modulov prišla na rad otázka, akým spôsobom budú spúšťané tieto funkčné moduly. Ideálne je, aby boli spúšťané pravidelne v určitých časových intervaloch, a aby tak dáta v PIS QAM boli stále, čo do najväčšej miery aktuálne vzhľadom na SAP ERP. Používateľ a sme tým samozrejme nemohli zaťažovať, keďže niektoré funkčné moduly by mali byť vyvolávané aj každých 15 minút.

Rozhodli sme sa preto využiť tzv. plánované joby bežiace na pozadí (scheduled background jobs), ktoré nám umožňujú pravidelné spúšťanie nami zvolených úloh v určitých časových intervaloch. V biznis vrstve sme si vytvorili samostatnú triedu pre každý plánovaný job. V triede je implementovaná metóda, ktorá je označená anotáciou

@Schedule, pomocou ktorej vieme nastaviť presný čas, kedy má byť metóda vyvolávaná, alebo iný určitý časový interval. V rámci takto vytvoreného plánovaného jobu vyvoláme konkrétny funkčný modul a vykonáme všetky potrebné operácie, ako je vytvorenie či aktualizácia dát v PIS QAM. Z plánovaného jobu nepristupujeme k funkčným modulom priamo, ale cez triedu SapService, ktorá sa tiež nachádza v biznis vrstve. Táto trieda riadi komunikáciu so SAP modulom prostredníctvom jeho rozhrania SapInterface.

3.19 DOKUMENTOVANIE SYSTÉMU

Z PIS QAM sa inkrementálnym vývojom za účelom splnenia požiadaviek stal rozsiahly systém, ktorého naštudovanie si samotnej implementácie si môže pre programátorov vyžadovať čas väčší než len pár minút. Z tohto, ale aj iných dôvodov je dôležité, aby bol PIS QAM správne dokumentovaný.

Pri dokumentovaní kódu sme využívali Javadoc. Ide o dokumentačné komentáre začínajúce trojicou znakou /**. Tieto komentáre sa pridávajú tesne pred oblasť, ktorú chceme zdokumentovať. Takto môžeme zdokumentovať rozhranie alebo triedu, jej atribúty, metódy, ako aj význam ich parametrov. Využívajú sa na to rôzne definované značky, ako napríklad @author definujúca autora, @param popisujúca parameter, či @returns, ktorá popisuje návratovú hodnotu metódy. Aj pomocou týchto značiek poskytujeme informácie pre lepšie formátovanie vytváranej dokumentácie. Totižto programátorovi sú tieto dokumentačné komentáre k dispozícii nielen priamo v kóde, ale je možné z nich vygenerovať aj html dokument (Pecinovský, 2012).

Súčasťou dokumentácie PIS QAM sú aj UML diagramy. Prvé verzie diagramov vznikli počas návrhu systému. S vývojom PIS QAM sa menili Java triedy, ale aj databáza, čím sa pôvodný diagram tried a ERD diagram stali neaktuálnymi. Niekoľko krát sme ich preto počas vývoja aktualizovali. ERD diagram bolo možné bez nejakých väčších problémov upraviť aj manuálne, no o diagrame tried sa to už povedať nedá. Vzhľadom na veľkosť PIS QAM by bolo náročné prehľadávať celý PIS QAM, porovnávať ho s existujúcim diagramom a zmeny zaznamenávať. Riešenie sme našli v reverznom inžinierstve, ktoré je poskytované aj nástrojom Enterprise Architect 10. Po jeho konfigurácii dokázal Enterprise Architect 10 vygenerovať niekoľko diagramov tried, ktoré sme ešte po pár manuálnych úpravách mohli použiť pre dokumentovanie systému.

4 VÝSLEDKY RIEŠENIA A ICH ZHODNOTENIE

V nasledujúcich riadkoch zhodnotíme PIS QAM, jeho význam a využitie, ale aj postupy, pre ktoré sme sa rozhodli pri jeho vývoji. Tiež zhodnotíme aj mieru naplnenia požiadaviek, ktoré sme si stanovili na začiatku vývoja. Prvé skúsenosti a získané výsledky sme publikovali aj v článku Automatizácia podnikových procesov v manažérstve kvality (Tóth, 2016).

Pri vývoji PIS QAM sme dbali na to, aby vyvíjaný informačný systém mal jednotlivé charakteristické vlastnosti informačných systémov, ktoré sme uviedli v kapitole 1.2.2 Informačný systém:

- *strategická orientácia* – PIS QAM podporuje dosiahnutie strategických cieľov podniku, pričom sa zameriava najmä na podporu manažérstva kvality v podniku, vykonávaných procesov v tejto oblasti a ich zefektívnenie, či až zautomatizovanie; prínos môže byť preukázaný aj v lepšom organizovaní a plánovaní práce na oddelení zabezpečovania kvality v podniku,
- *adekvátne funkčné spektrum* – PIS QAM pokrýva požiadavky používateľov, ktoré boli postupne napĺňané inkrementálnym spôsobom vývoja a dodávané ako verzie PIS QAM,
- *relevantnosť* – PIS QAM rieši skutočné problémy pracoviska a poskytuje podporu pre už existujúce procesy na pracovisku,
- *dostupnosť* – PIS QAM je dostupný kedykoľvek a kdekoľvek v rámci intranetu vybranej spoločnosti; táto výhoda je zabezpečená samotnou implementáciou PIS QAM; PIS QAM je implementovaný ako webová aplikácia, ktorá je nasadená na serveri, na ktorý majú prístup všetky počítače a mobilné zariadenia pripojené do intranetu; používateľovi pre prácu s PIS QAM postačuje len webový prehliadač, čiže si nemusí nič navyše inštalovať vo svojom počítači; používateľovi (pracovníkovi oddelenia zabezpečovania kvality) je takto poskytnutý jednoduchý prístup k informáciám, ktoré potrebuje pre vykonávanie svojej práce,
- *prehľadnosť* – dáta sú v PIS QAM roztriedené a zobrazené takým spôsobom, aby bola zabezpečená ich prehľadnosť; z tohto dôvodu sme zákazky rozdelili do tabuliek podľa stavu, v akom sa práve nachádzajú a vytvorili sme aj obrazovky, ktoré sú určené pre špecifické činnosti, ako je práca so spomínanými zákazkami, či protokolmi, čo by malo tiež prispieť k prehľadnosti,

- *presnosť a zrozumiteľnosť* – jednoznačnosť chápania informácie sme zabezpečili použitím terminológie, ktorá je zaužívaná pracovníkmi v danom odbore; preklady textov zobrazených v GUI boli stanovené odborníkmi, či už profesionálnymi prekladateľmi, alebo vedením oddelenia zabezpečovania kvality zo štátu, v ktorom je daný jazyk používaný,
- *jednoduchá doplniteľnosť* – jednoduchú doplniteľnosť na softvérovej úrovni zabezpečuje zvolená architektúra PIS QAM, do ktorej môžeme pridávať alebo odoberať moduly; cez rozhranie modulu dokážeme jednoducho pristupovať k jednotlivým poskytovaným funkciám modulu a zároveň aj definovať nové, prípadne odstrániť už nepotrebné funkcie; preklady všetkých textov zobrazovaných v GUI máme uložené na jednom mieste v properties súboroch, do ktorých vieme jednoducho doplniť nový preklad alebo v nich preklad aktualizovať, pričom nám stačí zmenu vykonať iba na tomto jednom mieste a zmena sa prejaví v celom PIS QAM; podobný princíp sme sa snažili do najväčšej miery použiť v celom PIS QAM, čo by malo zjednodušiť jeho ďalší vývoj a údržbu; aj pri poskytovaní dát, s ktorými sa v PIS QAM pracuje, je z pohľadu používateľa zabezpečená ich jednoduchá doplniteľnosť; dáta sú do PIS QAM pridávané a získavané automatizovane z rôznych zdrojov (SAP, Yocton) a to bez akéhokoľvek nutného zásahu používateľa; jedine pri získavaní dát zo súboru programu Yocton, je potrebné, aby používateľ tento súbor do PIS QAM nahral; automatizovane prebieha aj aktualizácia dát.

V priebehu vývoja PIS QAM sme sa stretli s rýchlo sa meniacim prostredím v podniku. Nestálosť prostredia a aj zmeny požiadaviek sme predpokladali, preto sa zvolenie si agilnej vývojovej metodiky Scrum prejavilo ako správne rozhodnutie.

Vyskytli sa aj situácie, kedy boli požiadavky niekoľkokrát menené a to už aj v čase, keď sme mali implementované funkcionality, ktorých sa zmeny dotýkali. Nepresnosti v požiadavkách boli spôsobené aj miernymi rozdielmi v procesoch jednotlivých podnikov vybranej spoločnosti. Zavedením PIS QAM do prevádzky sa systém vykonávania procesov štandardizuje a zosúladí pre všetky podniky danej spoločnosti.

Systém sa zaviedol aj do práce s dokumentmi a protokolmi. Pred zavedením PIS QAM do prevádzky ich mali pracovníci rôzne pozakladané a poukladané v počítačoch. Neprehľadnosť v protokoloch a dokumentoch mohla viesť k rôznym omylom, či nepresnostiam. Pracovníci majú teraz všetky protokoly a dokumenty uložené na jednom

mieste – v PIS QAM, kde sú ľahko dostupné a vyhľadateľné. Tým sa šetrí čas a urýchľujú sa procesy. Značné šetrenie času a zvýšenie efektivity prináša aj automatizovaná tvorba protokolov PIS QAM, čím sú pracovníci od tejto činnosti odbremenení. Pred zavedením PIS QAM do prevádzky museli pracovníci manuálne vytvárať protokoly a zapisovať do nich všetky údaje. V súčasnosti je protokol vytvorený automatizovane, pričom sú v ňom automaticky vypísané všetky potrebné údaje. Používateľovi do takto vytvoreného protokolu už stačí vypísať iba namerané hodnoty pre jednotlivé merania. Aj táto činnosť bude v nasledujúcich verziách PIS QAM zabezpečená automatizáciou. Vytvorí sa prepojenie medzi PIS QAM a meracími nástrojmi. PIS QAM získa namerané dáta priamo z meracích nástrojov a aj tieto dáta potom budú môcť byť zapísané do protokolu.

PIS QAM uľahčuje pracovníkom nielen administratívne činnosti, ale aj ich informuje, či varuje o potrebnosti vykonania niektorých činností v rámci ich procesov. Kvôli jasnej výstižnosti sme v PIS QAM použili ikony znázorňujúce svietiace diódy. Podľa farby akou svietia (zelená = v poriadku; žltá = skoro v poriadku; červená = varovanie), vedľa ľahko identifikovať, čo im v rámci ich procesov chýba a čo ešte musia urobiť, aby mohli zákazku úspešne ukončiť a mohla byť poslaná zákazníkovi. Takýto použitý spôsob informovania a varovania je jasný, výstižný, prehľadný a neprehliadnuteľný. Týka sa to napríklad procesov vytvárania protokolov, či overenia revízie.

Výhodou pre pracovníkov je aj sprehľadnenie a zorganizovanie práce so zákazkami. Vďaka tomu aj vedenie oddelenia zabezpečovania kvality dokáže lepšie plánovať a riadiť vykonávanie úloh. Všetky zákazky sú spoločne zobrazené a roztriedené podľa ich stavov. V tabuľkách sú prehľadne zobrazené najdôležitejšie údaje o zákazkách, pričom je umožnené zobrazenie aj detailu každej zákazky, kde používateľ vidí všetky potrebné informácie. Nie je tak nutné vyhľadávať informácie o zákazke v SAP ERP, kde to je o niečo zložitejšie.

PIS QAM v prvej verzii podporoval manuálne vytvorenie zákazky. Toto sa ukázalo ako nevýhodné vzhľadom na množstvo zákaziek a aj dát, ktoré bolo potrebné vyplniť. Vyžadovalo si to neprípustnú časovú náročnosť najmä z dlhodobého hľadiska. Vyriešili sme to automatizovaným získavaním dát a teda aj zákaziek z celopodnikového systému SAP ERP.

Pred samotným pripojením PIS QAM na SAP ERP za účelom získavania z neho dát, sme vykonali aj výkonnostné testovanie. Výkonnostný test PIS QAM sme najskôr

vykonali, keď bol PIS QAM v pôvodnom stave pred samotnou optimalizáciou, aby sme následne vedeli zhodnotiť účinnosť optimalizácie. Pri tomto prvom testovaní sa zistil závažný nedostatok a to pomalšie načítavanie stránok (obrazoviek GUI), čo je nežiadúce. Po optimalizácii Java kódu a aj po optimalizácii databázy sme pozorovali viditeľné zlepšenie v načítavaní stránok (čas načítavania sa skrátil viac ako desaťnásobne). Zistený nedostatok sme tým odstránili a poukázali sme na dôležitosť správnej optimalizácie softvéru, dotazov a aj databázy.

V rámci dokumentovania PIS QAM sme sa stretli s problémom tvorby diagramu tried. Aktualizácia manuálnou úpravou diagramu tried bola vzhľadom na rozsiahlosť PIS QAM náročná. Osvedčilo sa nám reverzné inžinierstvo ako riešenie pri tvorbe diagramu tried. Reverzné inžinierstvo a nástroj Enterprise Architect 10 nám uľahčilo prácu pri aktualizovaní diagramu tried. Enterprise Architect 10 po synchronizácii s kódom dokázal automatizovane vygenerovať všetky triedy, ich atribúty a metódy. To nám ušetrilo čas, keďže sme vďaka tomu nemuseli prehľadávať a porovnávať celý PIS QAM. Takto získaný diagram tried ale bolo potrebné ešte upraviť kvôli prehľadnosti. Museli sme upraviť rozmiestnenie vygenerovaných tried a vzťahov medzi nimi. Taktiež diagramy boli vygenerované po balíkoch, podľa štruktúry v akej máme uložené aj dané triedy. Ak sme chceli vytvoriť celkový diagram celého PIS QAM alebo jednotlivých modulov, museli sme vygenerované čiastkové diagramy zlúčiť.

Testovanie sme vykonali po implementácii každej novej funkcie. Pred nasadením na produkčný server sme PIS QAM najskôr nasadili na testovací server, kde prebehlo celkové otestovanie PIS QAM. Až následne bol PIS QAM nasadený na produkčný server a zavedený do prevádzky. Testovací a produkčný server boli dva samostatné aplikačné servery, ktoré sa ale nachádzali na jednom a tom istom fyzickom serveri. Okrem testovania výkonnosti a funkčnosti PIS QAM bola realizovaná aj statická kontrola kódu so zameraním na analýzu a meranie jeho kvality. Za týmto účelom sme využívali nástroj SonarQube. Vďaka tomuto nástroju aj vieme, že PIS QAM je tvorený 152 súbormi, 22457 riadkami (vrátane komentárov), 15952 riadkami kódu, z toho 13353 riadkov tvorí Java kód a 2599 riadkov tvorí kód webového GUI.

Zavedením PIS QAM do podnikov by sme mali dosiahnuť zefektívnenie interných podnikových procesov spojených s manažérstvom kvality, stanovenie jednotného systému pre podniky vybranej spoločnosti a skrátenie doby vykonávania procesov. PIS QAM je už takto úspešne zavedený do prevádzky a je používaný v dvoch podnikoch vybranej spoločnosti v Nemecku a v jednom podniku na Slovensku.

ZÁVER

V diplomovej práci sme sa zaoberali manažérstvom kvality a softvérovou podporou procesov prebiehajúcich v tejto oblasti, pričom sme využili podnikové informačné systémy. V teoretickej časti diplomovej práce sme uviedli význam manažérstva kvality v podnikoch pri riadení a zabezpečovaní kvality. Tiež sme popísali informačné systémy a spôsoby ich vývoja. V praktickej časti sme sa zamerali na procesy manažérstva kvality prebiehajúce v konkrétnych podnikoch vybranej spoločnosti. Uviedli sme prípadovú štúdiu manažérstva kvality v konkrétnom podniku. Pre procesy manažérstva kvality prebiehajúce v podnikoch sme vytvorili softvérovú podporu v podobe PIS QAM, ktorého vznik si vyžiadala nevyhovujúca situácia vo vybranej spoločnosti. PIS QAM je konkrétny výstup overený v praxi v prevádzke podnikov vybranej spoločnosti. Prispel k zefektívneniu, sprehľadneniu, automatizácii, lepšiemu plánovaniu a organizácii práce pracovníkov oddelenia zabezpečovania kvality. Prehľadne zobrazuje zákazky a všetky potrebné informácie pre pracovníkov, ktoré sú automatizovane získavané aj na základe pripojenia na SAP ERP. PIS QAM dokáže nielen automatizovane vytvárať protokoly, čím odbreňuje pracovníkov od tejto činnosti, ale dokáže pracovníkov aj informovať, či varovať na potrebné záležitosti, ktoré je ešte nevyhnutné spraviť, aby mohla byť zákazka uzavretá a odoslaná zákazníkovi.

Možnosti ďalšieho vylepšenia PIS QAM vidíme v zautomatizovaní získavania a zapisovania nameraných hodnôt z meraní vykonaných pri kontrole kvality výrobkov. Namerané hodnoty by mohli byť získavané priamo z meracích nástrojov, ktoré by boli prepojené s PIS QAM, uchovávali by sa v databáze a zapisovali by sa spolu s ostatnými dátami do protokolov pri ich automatizovanom vytváraní. V ďalšej verzii odporúčame implementovať aj používateľský manažment, s ktorým sa rátalo už v priebehu vývoja. Používatelia sa budú prihlasovať do PIS QAM a v logoch sa bude zaznamenávať vykonaná akcia spolu s menom používateľa. Navrhujeme aj implementovať zvyšné požiadavky, ktoré boli stanovené na začiatku vývoja, ako je vytvorenie funkcionality zadávania sťažností a definovania Tool Groups pracovníkom oddelenia výroby.

V diplomovej práci sme predstavili hlavné pracovné postupy, kľúčové myšlienky vlastného riešenia, zhrnuli sme skúsenosti získané pri optimalizácii implementácie a práci so zvolenými nástrojmi a technológiami. PIS QAM je úspešne zavedený do prevádzky v troch podnikoch vybranej spoločnosti v Nemecku aj na Slovensku.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

- AGILEMANIFESTO.ORG. 2001. Principles behind the Agile Manifesto. [online]. [cit. 2016-01-29]. Dostupné na internete: <<http://www.agilemanifesto.org/principles.html>>.
- APACHE.ORG. 2016. What is Maven?. [online]. [cit. 2016-02-24]. Dostupné na internete: <<https://maven.apache.org/what-is-maven.html>>.
- ARLOW, J. – NEUSTADT, I. 2011. UML 2 a unifikovaný proces vývoje aplikací. Computer Press. 2011. 567 s. ISBN 978-80-251-1503-9.
- BRUCKNER, T. A KOL. 2012. Tvorba informačních systémů. Grada Publishing, a.s.. 360 s. ISBN 978-80-247-4153-6.
- CARMAN, J. 2002. Write once, persist anywhere. Implement a Data Access Object pattern framework. [online]. [cit. 2016-02-10]. Dostupné na internete: <<http://www.javaworld.com/article/2074052/design-patterns/write-once--persist-anywhere.html>>.
- ÇIVICI, Ç. 2015. PrimeFaces user guide 5.0. [online]. [cit. 2016-02-09]. Dostupné na internete: <http://www.primefaces.org/docs/guide/primefaces_user_guide_5_0.pdf>.
- ECLIPSE.ORG. 2008. Scrum. [online]. [cit. 2015-08-28]. Dostupné na internete: <<http://epf.eclipse.org/wikis/scrum/>>.
- FILOVÁ, E. – MENDROŠOVÁ, K. 2009. Využitie ERP systémov v logistickom reťazci. [online]. [cit. 2016-02-01]. Dostupné na internete: <<http://www.logistickymonitor.sk/en/images/prispevky/erp-systemy.pdf>>.
- GÁLA, L. – POUR, J. – ŠEDIVÁ, Z. 2015. Podniková informatika. Počítačové aplikácie v podnikovej a mezipodnikovej praxi. Grada publishing, a.s.. 2015. 240 s. ISBN 978-80-247-5457-4.
- GEEKYVIVEK.WORDPRESS.COM. 2015. Interceptors in CDI Java EE 7 tutorial. [online]. [cit. 2016-02-15]. Dostupné na internete: <<https://geekyvivek.wordpress.com/2015/03/10/interceptors-jee-tutorial/>>.
- GONCALVES, A. 2013. Beginning Java EE 7. Apress. 2013. 573 s. ISBN: 978-1-4302-4627-5.

- HARAUSOVÁ, H. 2012. Procesné prístupy v manažerstve kvality. Prešovská univerzita v Prešove. 2012. 98 s. ISBN 978-80-555-0546-6.
- HENDERSON, K. 2000. Mistrovství v Transact-SQL. Computer Press. 2000. 496 s. ISBN: 80-7226-393-5.
- HRUBEC, J. 2001. Riadenie kvality. Slovenská poľnohospodárska univerzita v Nitre. 2001. 203 s. ISBN 80-7137-849-6.
- HVORECKÝ, J. 2013. Databázové technológie. Equilibria, s.r.o.. 2013. 330 s. ISBN 978-80-8143-082-4.
- INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. 2015. ISO 9000:2015 Quality management systems – Fundamentals and vocabulary.. [online]. [cit. 2016-03-02]. Dostupné na internete: <<https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en>>.
- INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Certification. [online]. [cit. 2015-08-13]. Dostupné na internete: <<http://www.iso.org/iso/home/standards/certification.htm>>.
- JARUNEK, P. 2014. SAP – Čo je to vlastne ten „SAP“? – časť 1. [online]. [cit. 2016-02-05]. Dostupné na internete: <<http://www.projectik.eu/index.php/programovanie/sap/item/247-co-je-sap>>.
- JENDROCK, E. A KOL. 2014. Java Platform, Enterprise Edition: The Java EE Tutorial. [online]. [cit. 2016-02-10]. Dostupné na internete: <<https://docs.oracle.com/javaee/7/tutorial/index.html>>.
- KANISOVÁ, H. – MÜLLER, M. 2004. UML srozumiteľne. Computer Press. 2004. 158 s. ISBN 80-251-0231-9.
- KLIMEŠ, C. – DRLÍK, M. 2007. Informačné systémy. Univerzita Konštantína Filozofa v Nitre. 2007. 224 s. ISBN 978-80-8094-115-4.
- KOŠOVSKÁ, I. – VÁRYOVÁ, I. – FERENCZI VAŇOVÁ, A. 2013. ERP systémy a ich prínos pre účtovné a finančné oddelenie podniku. [online]. [cit. 2016-02-01]. Dostupné na internete: <http://www.tvp.zcu.cz/cd/2013/PDF_sbornik/31.pdf>.

- KOVÁČ, M. 2009. Trendy vývoja ERP systémov. [online]. [cit. 2016-02-01]. Dostupné na internete: <<http://www.sjf.tuke.sk/transferinovacii/pages/archiv/transfer/14-2009/pdf/234-236.pdf>>.
- KUČERA, M. – LÁTEČKOVÁ, A. 2004. Podnikové informačné systémy. Slovenská poľnohospodárska univerzita v Nitre. 2004. 209 s. ISBN 80-8069-452-4.
- KUČERA, M. – ŠTEFÁNEK, J. – CVEČKO, J. 2002. Informačné systémy v poľnohospodárstve. Slovenská poľnohospodárska univerzita v Nitre. 2002. 238 s. ISBN 80-8069-084-7.
- MARIAŠ, M. 2000. Informačné systémy marketingu. Ekonom. 2000. 157 s. ISBN 80-225-1345-8.
- MATEIDES, A. – STYK, O. – KUČERA, M. 1998. Od kontroly cez ISO 9000 k TQM. Epos. 319 s. ISBN 80-8057-094-9.
- MOLNÁR, Z. 2001. Efektivnost informačních systémů. Grada Publishing. 2001. 179 s. ISBN 80-247-0087-5.
- MOLNÁR, Z. 2001. Podnikové informační systémy. [online]. [cit. 2015-07-31]. Dostupné na internete: <broucek.chytrak.cz/download/fsi/PIS/PIS1.ppt>.
- MUKHAR, K. A KOL. 2006. Beginning Java EE 5: From Novice to Professional. Apress. 2006. 641 s. ISBN 1-59059-470-3.
- ORACLE.COM. 2015. Java(TM) EE 7 Specification APIs. [online]. [cit. 2016-02-10]. Dostupné na internete: <<https://docs.oracle.com/javaee/7/api/toc.htm>>.
- ORACLE.COM. Core J2EE Patterns – Data Access Object. [online]. [cit. 2016-02-10]. Dostupné na internete: <<http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>>.
- ORACLE.COM. Overview of Interceptors. [online]. [cit. 2016-02-15]. Dostupné na internete: <<https://docs.oracle.com/javaee/6/tutorial/doc/gkigq.html>>.
- PANORAMA CONSULTING SOLUTIONS. 2014. 2014 ERP report. [online]. [cit. 2016-02-01]. Dostupné na internete: <<https://partners.megaventory.com/assets/2014-ERP-Report-2.pdf>>.
- PAULOVÁ, I. 2013. Komplexné manažérstvo kvality. Iura Edition. 2013. 160 s. ISBN 978-80-8078-574-1.

- PECINOVSKÝ, R. 2012. Java 7 učebnice objektové architektury pro začátečníky. Grada Publishing, a.s.. 2012. 496 s. ISBN 978-80-247-3665-5.
- RÁBOVÁ, I. – DRLÍK, M. 2013. Podnikové informačné systémy, ich architektúra a vývoj. Univerzita Konštantína Filozofa v Nitre. 2013. 168 s. ISBN 978-80-558-0287-9.
- SAP. 2014. SAP Java Connector. [online]. [cit. 2016-02-19]. Dostupné na internete: <https://help.sap.com/saphelp_nwpi711/helpdata/en/48/70792c872c1b5ae10000000a42189c/content.htm>.
- SOMMERVILLE, I. 2011. Softwarové inženýrství. Computer Press. 2013. 680 s. ISBN 978-80-251-3826-7.
- ŠATANOVÁ, A. – GEJDOŠ, P. 2007. Riadenie kvality. Technická univerzita vo Zvolene. 2007. 110 s. ISBN 978-80-228-1753-0.
- TÓTH, T. 2016. Automatizácia podnikových procesov v manažérstve kvality. In Študentská vedecká konferencia 2016. Zborník recenzovaných príspevkov (v tlači). Nitra, Slovenská republika.
- ÚRAD PRE NORMALIZÁCIU, METROLÓGIU A SKÚŠOBNÍCTVO SLOVENSKEJ REPULIKY. Posudzovanie zhody. [online]. [cit. 2015-08-13]. Dostupné na internete: <<http://www.unms.sk/?posudzovanie-zhody>>.
- ÚRAD PRE NORMALIZÁCIU, METROLÓGIU A SKÚŠOBNÍCTVO SLOVENSKEJ REPULIKY. Nové vydanie STN EN ISO 9000, 9001, 14001. [online]. [cit. 2015-03-02]. Dostupné na internete: <<https://www.sutn.sk/news.aspx?id=25>>.
- VASILIEV, Y. 2008. Beginning Database-Driven Application Development in Java™ EE: Using GlassFish™. Apress. 2008. 573 s. ISBN: 978-1-4302-0964-5.

ZOZNAM PRÍLOH

Príloha A – DVD nosič s diplomovou prácou vo formáte PDF

Príloha B – ERD diagram

Príloha C – Diagram tried

Príloha D – Ukážka obrazoviek PIS QAM