TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

KONTROLA KVALITY PROGRAMOVACÍHO KÓDU V JAZYCE ABAP

Bakalářská práce

Studijní program: B6209 – Systémové inženýrství a informatika
Studijní obor: 6209R021 – Manažerská informatika

Autor práce: **Tomáš Karhut**
Vedoucí práce: Mgr. Tomáš Žižka

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Economics

CHECK OF PROGRAMMING CODE QUALITY IN ABAP LANGUAGE

Bachelor thesis

Study programme: B6209 – System Engineering and Informatics
Study branch: 6209R021 – Managerial Informatics

Author: **Tomáš Karhut**
Supervisor: Mgr. Tomáš Žižka

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta
Akademický rok: **2014/2015**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Karhut**
Osobní číslo:	**E12000471**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Manažerská informatika**
Název tématu:	**Kontrola kvality programovacího kódu v jazyce ABAP**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

1. Přístupy k automatické kontrole kvality programovacího kódu
2. Vývoj dodatků či nových metod pro automatický průběh kontroly
3. Specifika spolupráce s interními týmy vývojářů v SAP
4. Stanovení procedury a automatického vyhodnocení nového vývoje
5. Zhodnocení navrženého řešení

Rozsah grafických prací:

Rozsah pracovní zprávy: **30 normostran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

KÜHNHAUSER, Karl-Heinz. ABAP Výukový kurz. 1. vyd. Brno: Computer press, a.s., 2012. ISBN 978-80-251-2117-7.
MASSEN, André, Markus SCHOENEN, Detlev FRICK, Andreas GADATSCH. SAP R/3 Kompletní průvodce. 1. vyd. Brno: Computer Press, a.s., 2007. ISBN 978-80-251-17750-7.
MEIJS, Ben, Albert KROUWELS, Wouter HEUVELMANS, Ron SOMMEN. Enhancing the Quality of ABAP Development. 1st. ed. Walldorf: Galileo Press, 2004. ISBN 1-59229-030-2.
THOMASIS, Tony de, Alisdair TEMPLETON. Managing Custom Code in SAP. 1st ed. Boston: Galileo Press, 2013. ISBN 978-1-59229-436-7.
Elektronická databáze článků ProQuest (knihovna.tul.cz).

Vedoucí bakalářské práce: **Mgr. Tomáš Žižka**
Katedra informatiky

Konzultant bakalářské práce: **Ing. Václav Němec**

Datum zadání bakalářské práce: **31. října 2014**

Termín odevzdání bakalářské práce: **7. května 2015**

doc. Ing. Miroslav Žižka, Ph.D.
děkan

L.S.

Technická univerzita v Liberci
Ekonomická fakulta

doc. Ing. Jan Skrbek, Dr.
vedoucí katedry

V Liberci dne 31. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto pří- padě má TUL právo ode mne požadovat úhradu nákladů, které vyna- ložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Anotace

Předmětem této bakalářské práce je snaha optimalizovat proces kontroly kvality programovacího kódu v jazyce ABAP. Práce je rozdělena do dvou částí, teoretická část popisuje již existující a používané přístupy k automatické kontrole programovacího kódu a také nově vyvíjené metody.

Praktická část analyzuje postupy používané týmem pracujícím v systému SAP ve společnosti ŠKODA AUTO a. s. Popisuje spolupráci s interními týmy vývojářů SAP a následné vyhodnocení stávající situace v oblasti kontroly programovacího kódu.

Cílem této práce je přiblížit postupy a procesy, které se používají ke kontrole programovacího kódu v jazyce ABAP v oddělení EOA společnosti Škoda auto a navrhnout způsob, jak tyto procesy zdokonalit.

Klíčová slova

jazyk ABAP, kontrola kvality, programovací kód, systém SAP

Annotation

The subject of this thesis is an effort to optimize the process of quality check of programming code in ABAP language. The thesis is divided into 2 parts, theoretical part describes existing approaches to automatic quality check of programming code and new methods in development.

The practical part analyses methods used by the team working in SAP in SKODA AUTO a. s. It describes cooperation with internal teams of developers in SAP and evaluation of the current situation in the area of programming code check.

The aim of this work is to describe methods and procedures that are used in the EOA department of Skoda auto to check programming code in ABAP language and to propose a way to perfect those procedures.

Key Words

ABAP language, quality check, programming code, SAP system

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Dané téma jsem si zvolil zejména proto, že úzce souvisí z aktivitami, které na praxi ve Škoda auto vykonávám. Dalším důvodem bylo doporučení tohoto téma konzultantem mé bakalářské práce panem Ing. Václavem Němcem. Protože jsem na praxi v IT oddělení, setkávám se zde s programovacím kódem běžně, stejně jako velká většina zaměstnanců, kteří zde pracují. Je tedy důležité, aby byl program napsán co nejkvalitněji a nejefektivněji.

Cílem této bakalářské práce je popsat procesy používané ke kontrole kvality programovacího kódu v systému SAP ve Škoda auto a navrhnout způsoby, jakými by se dal stávající proces zdokonalit.

První část práce bude zaměřena především na popis procesů kontroly programovacího kódu v jazyce *Advanced Business Application Programming* (dále jen ABAP) používaných ve Škoda auto i zbytku koncernu *Volkswagen* (dále jen VW), který naše oddělení podporuje, budou zde však zmíněné i obecné přístupy používané v jiných programovacích jazycích.

Analytická část práce nabízí detailnější pohled na proces kontroly v jazyce ABAP ve Škoda auto, ze zkušeností získaných při spolupráci s vývojáři u konkrétních programů a kódů při jejich kontrole.

Literární rešerše

Michael Fagan, autor takzvané „Faganovy inspekce," (strukturovaný proces s cílem nalézt chyby ve vývoji) ve své knize *Design and Code Ispections to Reduce Errors in Program Development* říká, že úspěšné řízení jakéhokoliv projektu vyžaduje plánování, měření a kontrolu. V prostředí vývoje programů se tyto požadavky mění na potřebu definovat programovací proces z hlediska souboru operací, kdy každá operace má svoje vlastní výstupní kritéria. Dále musí existovat nějaký způsob, jak změřit kompletnost produktu v jakémkoliv bodě jeho vývoje pomocí inspekce nebo testování. Takto změřená data pak mají být použita pro kontrolu celého procesu.

Florian Deissenboeck, spoluzakladatel firmy CQSE GmbH ve své publikaci s názvem *Tool Support for Continous Quality Control* zastává ten názor, že postupem času softwarové systémy ztrácí na kvalitě a náklady na jejich údržbu mohou růst, pokud organizace nepodnikne preventivní opatření. Kontrola kvality je prvním krokem k zabránění růst nákladů. Dlouhodobé měření kvality kódu pomáhají uživatelům identifikovat problémy s kvalitou včas, dokud jejich náprava ještě není příliš nákladná. Dlouhodobá a integrovaná zpětná vazba také pomáhá vývojářům zdokonalit jejich dovednosti a tím snižují pravděpodobnost, že se v budoucnu vyskytnou nedostatky v kvalitě vývoje. Aby byla možná pravidelná kontrola kvality, musí být do značné míry automatizovaná a výsledky musejí být vhodně zpracovány, aby nezahltily uživatele příliš velkým množstvím dat.

Richard Bache a **Monika Müllerburg** ve své knize *Measures of testability as a basis for quality assurance* říkají, že testování programu je nejpoužívanější způsob pro zajištění kvality kódu. V průběhu životního cyklu programu je velké množství času a úsilí věnováno testování programu. Je užitečné mít způsob, jak usnadnit organizaci procesu vývoje programu a jeho testování. Proto je množství úsilí věnovanému testování programu z hlediska kvality důležitý atribut, který se dá nazvat jako testovatelnost.

1 Kvalita programovacího kódu

Termín kvalita programovacího kódu v prostředí ABAP neznamená pouze schopnost programu vyhovět požadavkům klienta, takto by se dal za kvalitní považovat jakýkoliv program, který alespoň částečně plní funkci kterou má. Dobře napsaný program by měl také dosahovat určité úrovně technické kvality. [1]

1.1 Technická kvalita

Technickou kvalitou se rozumí rozdíl mezi rychle napsaným odbitým programem a pečlivě napsaným propracovaným programem. Důležitost technické kvality při vývoji v prostředí ABAP je všeobecně podceňována.

Dobře vyvinutý program nesplňuje pouze očekávání uživatelů, nýbrž také zajišťuje, že požadovaný výstup programu je vždy správný. Takovýto program by měl být odolný vůči nesprávně zadaným datům nebo použití v nevhodné situaci. Neměl by být ani ovlivněn změnami a novými implementacemi v rámci systému SAP. Práce s programem z hlediska uživatele by měla být intuitivní a snadná v jakékoliv situaci. V neposlední řadě patří mezi vlastnosti technicky vysoce kvalitního programu také možnost snadné změny kódu v případě potřeby, malá změna by měla vyžadovat pouze přepis několika řádků kódu a minimální úsilí. [1]

1.2 Prvky technické kvality

Pro přesnější a detailnější rozbor technické kvality programu je rozlišováno několik prvků, patří mezi ně korektnost, stabilita, uživatelské prostředí, účinnost a udržovatelnost. Pro jejich důležitost jsou tyto prvky detailněji rozepsány níže. [1]

1.2.1 Korektnost

Termín korektnost vyjadřuje kvalitu dat vyprodukovaných daným programem. Výstupní data vygenerovaná programem by měla být přesná a odpovídat tomu, jaká vstupní data byla do programu zadána. Korektnost bývá někdy rozdělována do dvou kategorií a to sice na totální a částečnou.

Totálně korektní program v případě, že by výstupní data měla být nesprávná nebo nepřesná, ihned ukončí svůj běh a vypíše chybovou hlášku. Částečně korektní program se nemusí ukončit, měl by však fungovat tak, že pokud nabídne výstupní data, měla by být vždy správná.

Při popisu termínu korektnost programu je třeba zmínit také robustnost. Robustní program je v zásadě opakem korektního. Dokonale robustní program by měl být napsán tak, že nedojde k jeho ukončení při výskytu jakékoliv chyby a bude pracovat dále. [1]

1.2.2 Uživatelské prostředí

Uživatelským prostředím se rozumí použitelnost programu z pohledu uživatele. Program by měl vypadat dobře na oko a orientace v něm by měla být intuitivní. Uživateli by nemělo dělat problémy navigovat se v rámci programu přes jednotlivé obrazovky a v případě potřeby by měl program nabídnout uživateli pomoc kdykoliv ji potřebuje. Podobně jako korektnost je nepřívětivé uživatelské prostředí snadné poznat již v prvních momentech používání programu. [1]

1.2.3 Účinnost

Účinnost programu je poněkud komplexnější výraz. Je totiž rozdíl v tom, o jaký typ programu se jedná. Program běžící na popředí by měl reagovat velmi rychle, v rámci desetin, maximálně několik sekund. Program pracující na pozadí může běžet daleko déle. Všechny programy jsou však ovlivněny specifikacemi stroje, na kterém běží, tedy jak výkonný má

hardware, jak rychlý je jeho operační systém a jaký používá databázový systém (z anglického DBMS – Database Management System). Kvalita programu z hlediska účinnosti je důležitá především kvůli systému Enterprise Resource Planning.

ERP je informační systém, který integruje a automatizuje velké množství procesů souvisejících s činnostmi podniku. Zejména se jedná o výrobu, logistiku, distribuci, správu majetku, prodej, fakturaci a účetnictví.

ERP využívá sdílených zdrojů, proto může větší množství neoptimálně pracujících programů z hlediska kvality účinnosti mít negativní vliv na celkový běh systému. Problémy s funkčností systému se navíc většinou neobjeví hned, ale až s přibývajícím množstvím dat, tedy v momentě kdy se tyto problémy začnou objevovat, bývají o to závažnější a hůře řešitelné. [1]

1.2.4 Stabilita

Pojem stabilita u programu znamená, do jaké míry program pracuje podle očekávání z technického hlediska. Na rozdíl od problémů s účinností se problémy vyplývající z nedostatečné stability programu mohou objevit kdykoliv, například ihned po jeho implementaci nebo taky po několika měsících používání.

Nestabilní program typicky přestane pracovat zcela náhle a spadne, vypíše chybovou hlášku nebo se prostě zastaví. Předejít problémům se stabilitou programu jde z velké části jednoduše napsáním správného a přesného kódu. To ale vyžaduje i správné porozumění technickému prostředí programu. [1]

1.2.5 Udržovatelnost

Udržovatelnost programu vyjadřuje, do jaké míry lze aplikovat změny do programu někým jiným, než je jeho původní autor. Udržovatelnost závisí na úrovni popisu a komentáře ke kódu od autora programu a také na jaké úrovni je část kódu znovupoužitelná.

Naprostá většina programů je v průběhu své existence upravována, velmi často někým jiným než autorem samotným. K úpravám dochází často až po roce používání programu, nicméně jelikož dříve nebo později bude téměř každý program podroben údržbě, neměl by tento prvek technické kvality programu zůstat opomíjen. [1]

2 SAP

Abychom si mohli představit a popsat jednotlivé metody kontroly kvality programovacího kódu v jazyce ABAP, musíme nejdřív znát alespoň základy o samotném Systému SAP. Jazyk ABAP totiž existuje pouze v rámci systému SAP, nikoliv samostatně.

2.1 Historie systému SAP

SAP byl vyvinut stejnojmennou německou firmou, založenou v roce 1972 pěti bývalými zaměstnanci firmy IBM. Zkratka SAP znamená Systém, aplikace a produkty (z německého Systeme, Andwendungen und Produkte in der Datenverarbeitung). Tato firma měla za cíl vyvinout standartní software pro řízení podnikové ekonomiky. Již o rok později, tedy roku 1973, byl dokončen vývoj prvního standartního softwaru pro oblast finančního účetnictví. Tento software tvořil základ systému SAP R/1.

Jeho následovníkem byl systém SAP R/2, který je označován jako první systém ERP (Enterprise resource planning). Oproti svému předchůdci byl značně rozšířen, jeho provoz však stále vyžadoval použití sálových počítačů.

V roce 1992 vydala firma SAP další verzi svého systému s názvem SAP R/3. Oproti svým předchůdcům se jedná o zcela jiný produkt, přepracovaný téměř ve všech ohledech. SAP R/3 je již založen na architektuře klient-server a využívá relačních databází. Je také upraven tak, aby bylo možné jej provozovat na hardwaru od různých výrobců a aby běžel na počítačích s různými operačními systémy. Díky úspěchu tohoto systému dosáhla firma SAP vedoucího postavená na celosvětovém trhu se standartními softwary pro řízení podnikové ekonomiky.

Dalším technologickým skokem, který firma SAP učinila, bylo uvedení systému SAP R/3 Enterprise v roce 2002. Stávající základní systém byl nahrazen produktem SAP WebAS (SAP Web Application Server). Co se týče funkčnosti, však k žádným zásadním změnám

nedošlo. Jednotlivé moduly byly přeorganizovány a uspořádány novým způsobem, který umožnil vývoj některých rozšíření pro systém SAP. [2]

2.2 Skupina produktů SAP

SAP není pouze jediný systém, jedná se spíše o celou skupinu produktů, která představuje kompletní řešení nejen pro interní oddělení podniku. Pokrývá také všechny procesy nad rámec podniku. Dá se tedy říct, že některé komponenty přesahují rámec běžných systému ERP.

Celá skupina je rozdělena do tří oblastí. První oblast je tvořena technologickými komponenty, které jsou souhrnně označovány jako SAP NetWeaver. Druhá oblast obsahuje všechny komponenty týkající se řízení ekonomiky podniku. Tyto komponenty jsou souhrnně označovány jako xApps (Extended Applications). Třetí oblastí jsou průmyslová řešení (mySAP Business Suite), tento pojem znamená speciální dodatková řešení, používaná pouze v určitém odvětví. Takovýchto řešení vyvinula firma SAP v průběhu let desítky. [2]

2.3 Architektura systému SAP

Systém SAP používaný ve Škoda auto, tedy SAP R/3, je dále rozdělen dle takzvané 3-vrstvé architektury. Tyto tři vrstvy jsou:

- Prezenční vrstva
    - Tato vrstva slouží uživatelům pro vkládání vstupních dat a prezentuje jim výstupní data. To je umožněno pomocí *SAP Graphical user interface* (dále jen GUI), v překladu grafické uživatelské prostředí. SAP GUI je nainstalován na jednotlivých počítačích, které tak slouží jako prezenční vrstva. Představit si pod ním můžeme klienta systému SAP, který si uživatel spustí a pak v něm vytvoří požadavky, které se

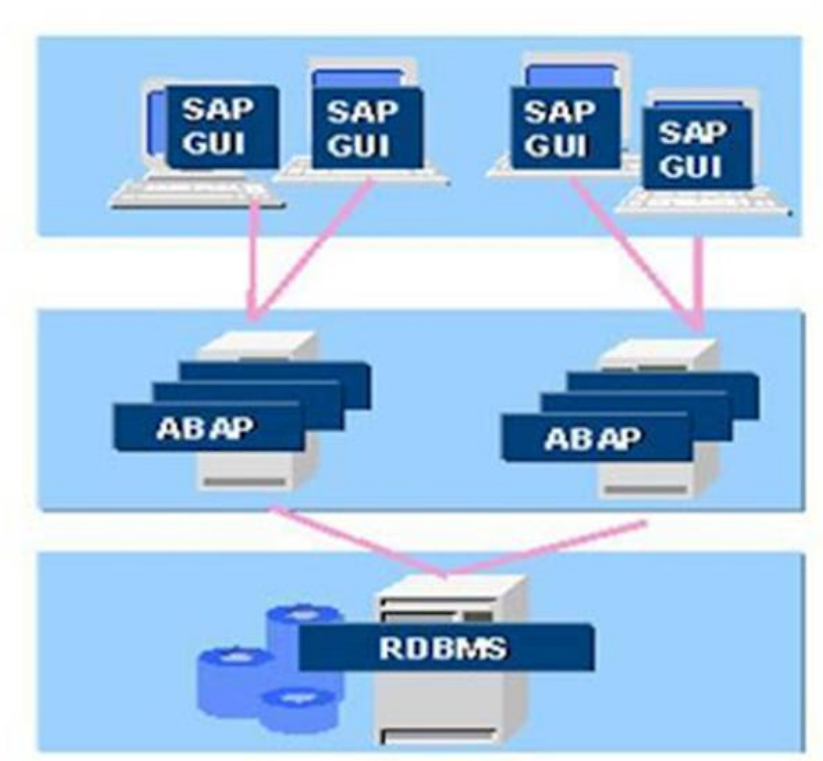
odešlou na aplikační vrstvu ke zpracování. Po zpracování pošle aplikační vrstva výsledky zpět SAP GUI, které je zformátuje a následně prezentuje výstup uživateli. [

- Aplikační vrstva

    - Tuto vrstvu tvoří soubor příkazů, které kolektivně interpretují ABAP programy a zpracovává pro ně vstupní a výstupní data. Když dojde ke spuštění této vrstvy, spustí se s ní zároveň i všechny příkazy. Stejně tak je to i při ukončení této vrstvy. Množství procesů, které proběhnou při spuštění této vrstvy je definováno v konfiguračním souboru s názvem *Application server profile*. Všechny ABAP programy běží na aplikační vrstvě, příkaz sice může být spuštěn z prezenční vrstvy, ale nemůže zde proběhnout ABAP program. V případě, že program vyžaduje informaci z databáze, aplikační vrstva definuje požadavek a vyšle ho na databázovou vrstvu.

- Databázová vrstva

    - Tato vrstva je tvořena souborem příkazů, které přijímají požadavky z aplikačního serveru. Tyto požadavky jsou předány na *Relation Database Management System* (dále jen RDMS), v překladu relační systém pro správu databáze. Samotný systém SAP totiž žádnou databázi neobsahuje, databázová vrstva tak pouze vezme data ze systému RDMS a předá je aplikační vrstvě, kde s těmito daty může dále pracovat ABAP program. Databázová vrstva obvykle běží na jediném serveru, na kterém může být i systém RDMS. Někdy však běží i tento systém na vlastním serveru. [3]



Obrázek 1: SAP R/3 Architektura
Zdroj: Interní materiály společnosti ŠKODA AUTO a.s.

2.4 Struktura systému SAP

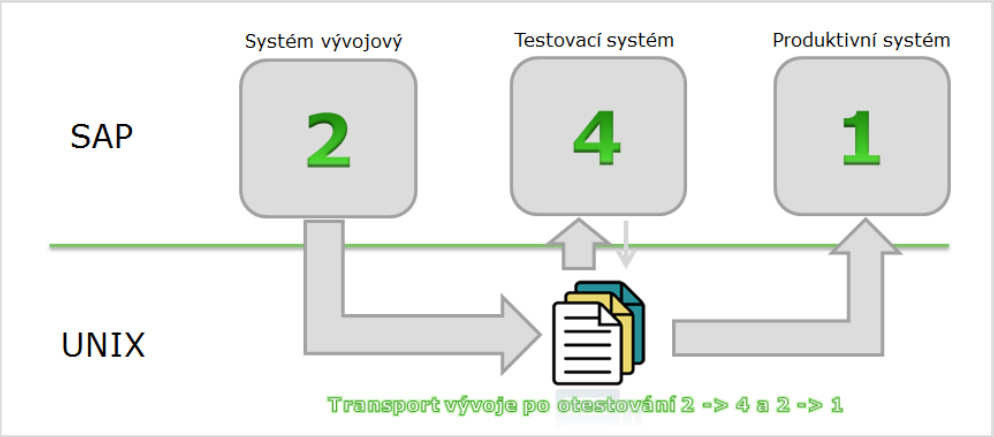
Struktura systému SAP R/3 se může lišit v závislosti na tom, co přesně od něj daná firma vyžaduje. Tato struktura je dána složením jednotlivých klientů a vztahy mezi nimi. Škoda auto využívá standartní strukturu doporučovanou vývojáři z firmy SAP pro velké společnosti, tedy tří systémovou strukturu. V této struktuře má každý z hlavních klientů svůj vlastní SAP systém. Tyto tři systémy jsou vývojový (označován číslem 2), testovací (4) a produktivní (1).

- Vývojový systém
    - V tomto systému probíhá veškerý vývoj.
- Testovací systém

- Nachází se mezi vývojovým a produktivním systémem. Tento systém slouží k tomu, aby se do produkce nedostala žádná nežádoucí data.

- Produktivní systém

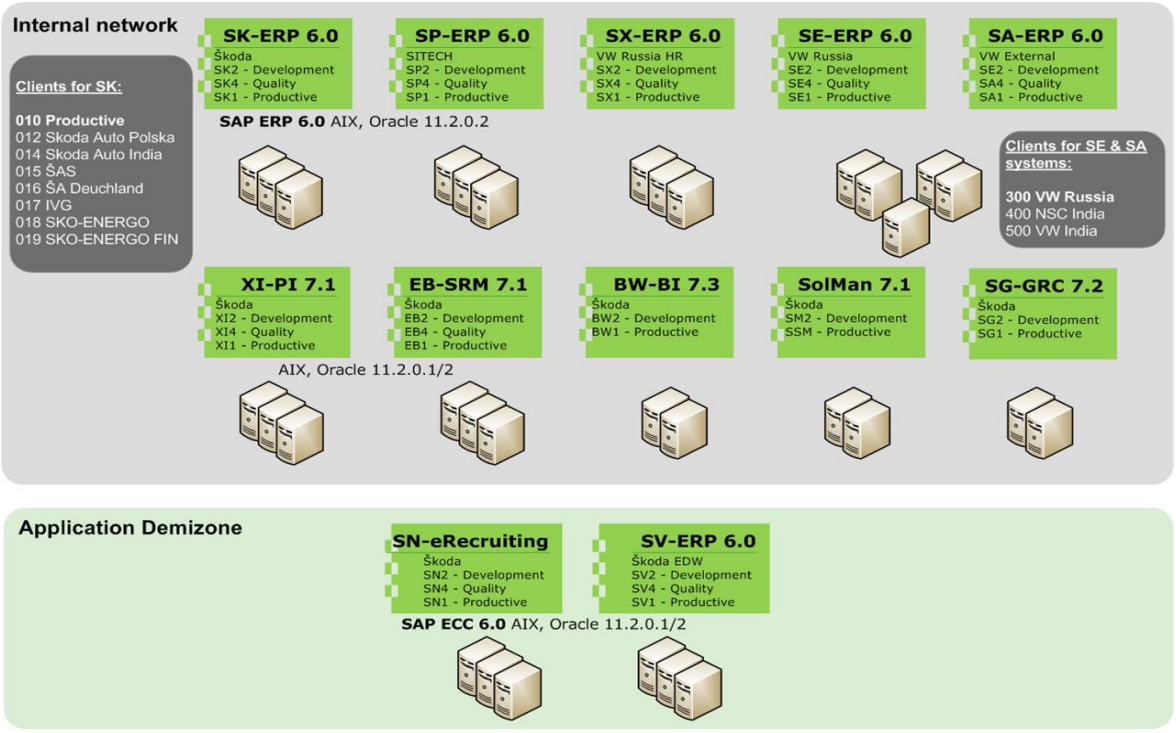
  - Slouží odborným útvarům. Používají ho koncoví uživatelé. [4]



Obrázek 2: SAP struktura
Zdroj: Interní materiály společnosti ŠKODA AUTO a.s.

2.5 Klienti systému SAP

Systém SAP se nedělí pouze na vývojový, testovací a produktivní, ale také na jednotlivé klienty, které slouží různým společnostem v rámci koncernu VW. Existují zde systémy pro společnosti, jako je Škoda auto v Polsku, Indii, Německu atd. Některé z těchto klientů mají pouze vývojový a produktivní systém, většina má ale všechny tři. Přehled těchto klientů můžeme vidět na obrázku níže. [4]



Obrázek 3 : Klienti SAP ve Škoda auto
Zdroj: Prezentace ke školení nováčků EOA

2.6 Transakce

V klientech systému SAP bylo dříve požadováno za standartní způsob pro spuštění nějakého programu vyhledání v nabídce. Systém však nabízí tak možnost přímého spuštění pomocí zadání technického názvu (kódu transakce neboli TCODE) požadované transakce do příkazové řádky. V dnešní době již naprostá většina zkušených uživatelů systému SAP používá právě kódy transakcí, které v této práci ještě budou zmíněny, zejména v souvislosti s jednotlivými programy. [1]

2.7 Transport

Veškeré změny udělená ve vývojovém systému SAP je třeba aplikovat do produktivního systému pomocí transportů. Transport je mechanizmus, sloužící k přesunutí konfigurace

z jednoho SAP systému do jiného. Zjednodušeně řečeno se jedná o skupinu úloh, které provádějí změny v tabulkách.

Transporty se dělí do 3 kategorií:

- Workbench transport
    - Týká se všeho vývoje, při kterém dochází ke změně programového kódu. Standardně nebývá klientově závislý (nadklientový).
- Customizing transport
    - Týká se všech nastavení (změny v tabulkách). Standardně bývá klientově závislý.
- Transport kopií
    - Při vývoji složitějších programů se do testovacího systému odešle nejprve kopie transportu. Originál transportu se pošle až po ukončení testů. [4]

2.7.1 Složení transportu

Transport se skládá z několika souborů, které jsou většinou umístěny v 6 oddělených složkách v rámci operačního systému.

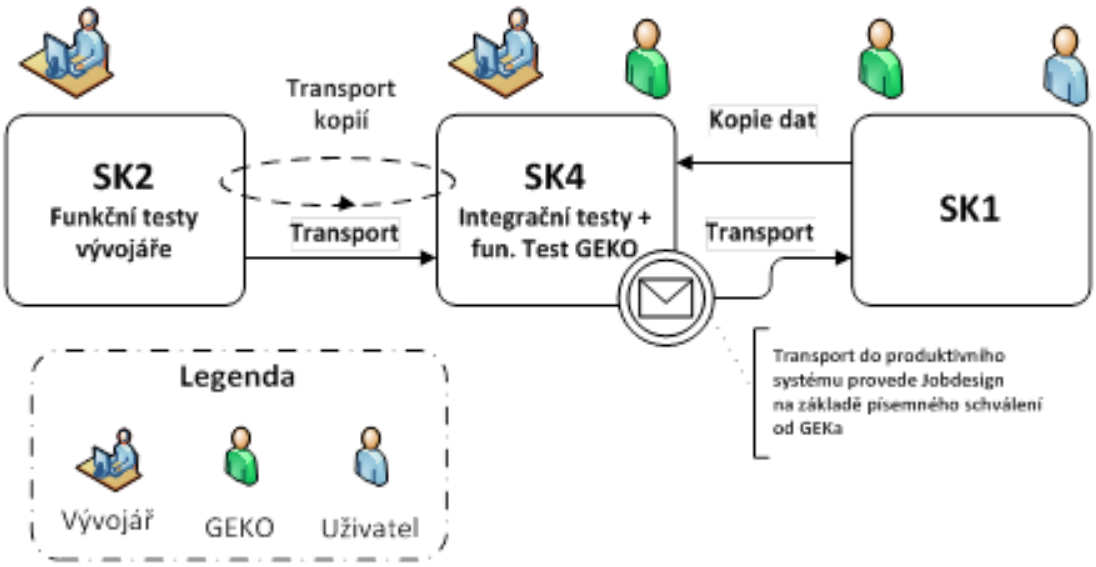
- Datové soubory
    - Soubory obsahující transportní data, tedy data, která mají být transportována.

- CO-soubory
    - Obsahují informace týkající se požadavků ke změně, tedy jednotlivé kroky požadavku ke změně.
- Profilové soubory
    - Obsahují informace o databázích z cílových systémů.
- Transport log
    - Obsahuje data týkající se historie transportu.
- Podpůrné balíčky
    - Obsahují soubory týkající se aktualizací a vylepšení.
- Binární složka
    - Obsahuje soubory sloužící ke konfiguraci TMS (Transport Management System) [4]

2.7.2 Uvolňování transportů

Jakmile je vývojář připraven transport exportovat do testovacího systému, musí ho uvolnit, což se provádí přímo v některém z klientů systému SAP pomocí transakce SE01 (Transport organizer). Poté vývojář osloví klíčového uživatele (GEKO) pro danou oblast, který má jako jediný právo žádat provedení přenosu transportu do produktivního systému. GEKO však nesmí zažádat o provedení přenosu daného transportu dřív, než si ověří, zda byly změny obsažené v transportu důkladně otestovány a jsou tak připraveny pro zavedení do produktivního systému.

Změny v transportech se testují v několika krocích, nejprve je přímo ve vývojovém systému testuje vývojář. Následně je vývojář otestuje i v testovacím systému, tento test se nazývá *Základní funkční test*. V případě, že změny projdou oběma těmito testy, putují zpět k zadavateli, který je otestuje pomocí „Integračního testu," který se provádí také v testovacím systému. Všechny tyto kroky musejí být také zdokumentovány jednotlivými účastníky procesu. Poté pracovníci z oddělení EOI/4, tedy JOBDESIGN provedou přenos transportu pomocí takzvaných jobů a změny obsažené v transportu se objeví v produktivním systému, ve kterém pracují koncoví uživatelé. [4]



Obrázek 4: Transport ve Škoda auto
Zdroj: Prezentace ke školení nováčků EOA

2.8 Jazyk ABAP

ABAP je programovacím jazykem integrovaným v systému SAP R/3. Jedná se o programovací jazyk čtvrté generace (4 GL). Tento pojem obecně znamená, že programy tohoto typu operují s velkým množstvím informací naráz. Tyto programy také často zahrnují podporu řízení databáze, generátor reportů, matematickou optimalizaci, vývoj GUI nebo webový vývoj. Jazyky 4 GL bývají často přirovnávány k doménově specifickým jazykům (DSLs), některé výzkumy uvádějí jazyky 4 GL jako sub typ DSLs. Tyto funkce mají navíc

oproti skupině jazyků třetí generace, do kterých patří známé a často používané jazyky jako je C, C++, C# nebo Java.

Jazyk ABAP nemůže existovat samostatně v prostředí nějakého operačního systému, jelikož k zajištění funkčnosti potřebuje celou řadu programů, které zprostředkovávají načtení, interpretaci a vyrovnávací paměť vstupů a výstupů z jednotlivých programů.

Jazyk ABAP se nachází v aplikační vrstvě v třívrstvé architektuře systému SAP R/3. Tato vrstva obsahuje softwarové komponenty potřebné ke spuštění programu včetně SAP kernel, což je jádro aplikace, které obsahuje spouštěcí soubory pro programy s příponou EXE. [5]

3 Vývojové prostředí ABAP

Vývojové prostředí jazyka ABAP (ABAP Development Workbench) je plně integrovaná sada vývojových pomůcek, slovníku dat a programovacího jazyka. Každý ABAP program má k sobě přidělený typ, který určuje, jak je program spouštěn a které zdroje program ke svému chodu využívá. Celkově je ve vývojovém prostředí ABAP rozlišováno 8 typů programů. Každý z těchto typů je označen písmenem a spojován s určitým klíčovým slovem. [5]

3.1 Typy programů v jazyce ABAP

Níže je jednotlivě popsáno všech 8 typů programů v jazyce ABAP.

3.1.1 Spustitelné programy

Programy tohoto typu se vyznačují klíčovým slovem Report a jsou označovány písmenem I. Jsou vyvíjeny pomocí transakce SE38, spustitelné přímo z vývojového prostředí ABAP. Spustitelné programy se taky velmi často používají v dalších programech, ze kterých mohou být zavolány několika způsoby. Hlavním účelem pro tento typ programů je zobrazovat velké množství dat. Příkazy jsou zde většinou spojené s určitými událostmi, jako je inicializace, spuštění výběrové obrazovky, start výběru, konec výběru atd. [6]

3.1.2 Modulové programy

Tento typ programů je spojován s klíčovým slovem Program a značí se písmenem M. Jsou také označovány jako dialogové programy, protože jsou používány ke správě sekvence několika po sobě jdoucích oken. Tyto programy se skládají ze 4 komponentů:

- Výběrové obrazovky

- Transakční kódy
- GUI status
- ABAP program [6]

3.1.3 Programy tříd

Typ programů označovaný písmenem K, spojený s klíčovým slovem Třída. Programy tohoto typu mohou obsahovat jednu globální třídu a jakýkoliv počet lokálních tříd. Nemohou však být spouštěny samostatně, jsou načítány pomocí jejích globální třídy. Jsou vyvíjeny pomocí funkce Class Builder. [6]

3.1.4 Programy s rozhraním

Tyto programy jsou značeny písmenem J a je k nim přiřazováno klíčové slovo Rozhraní. Podobně jako programy tříd obsahují pouze jedinou definici globálního rozhraní, která může být implementována do jakékoliv globální či lokální třídy. Také jsou vyvíjeny ve funkci Class Builder. [6]

3.1.5 Sub rutinní programy

Tento typ je vázán s klíčovým slovem Program a označován písmenem S. Jedná se o blok kódu nachazející se mezi příkazy FORM a ENDFORM. V zásadě se dělí na 2 podtypy.

- Interní
    - Tento podtyp sub rutinního programu je definován v tom samém programu, ze kterého je volán. Volající program má přístup ke všem datům a objektům, které jsou definované v sub rutinním programu.

- Externí
    - Druhý z podtypů je definován mimo program, ze kterého je volán. Podobně jako interní podtyp nabízí přístup k datům definovaným v běžných a globálních částech paměti. [6]

3.1.6 Funkční skupina

Programy patřící do tohoto typu jsou značeny písmenem F a jsou spojovány s klíčovým slovem Funkce. Nemohou být spouštěny samostatně, nýbrž pouze v rámci určitého funkčního modulu, do kterého spadají. Všechny funkční moduly spojené s danou funkční skupinou nabízí přístup ke všem globálním datům. Systém SAP podporuje velké množství standartních funkčních skupin. Tyto programy jsou vyvíjeny pomocí funkce Function Builder v transakci SE37. [6]

3.1.7 Include programy

Tento typ se stejně jako spustitelné programy označuje písmenem I, na rozdíl od nich je však spojován s klíčovým slovem Include. Tento typ je využíván velmi často, zejména z důvodu zkrácení délky programovacího kódu původního programu. Toto funguje tak, že předem vytvořená část kódu, tedy Include program, se vloží do požadovaného programu pomocí příkazu INCLUDE [Název programu] a tento program poté může pracovat s řádky z Include programu, jako kdyby je sám obsahoval.

Podobně jako jiné typy programů v jazyce ABAP i Include program nemůže být spouštěn samostatně, ale musejí být zavolány v rámci spustitelného programu. Programovací kód, který je součástí Include programu je vyřešen před generováním ve spustitelném programu. Po vygenerování obsahuje spustitelný program zdrojový kód všech Include programů, které jsou v něm zahrnuty. [6]

3.1.8 Typové skupiny

Poslední z typů programů v jazyce ABAP je vázán ke klíčovému slovnímu spojení Type-Pool. Nemohou obsahovat vlastní obrazovky nebo procesní bloky. Pomocí jejich globálních typů dat mohou být viditelné v jiných ABAP programech. Vytváří se pomocí funkce ABAP Dictionary. [6]

4 Metody kontroly

Způsobů jak automaticky kontrolovat kvalitu kódu je několik, liší se podle systému a programovacího jazyka. U systému SAP a v něm používaném jazyce ABAP je k dispozici množství metod ke kontrole kódu, které budou v této kapitole popsány. [7]

4.1 Code Inspector

Tato funkce slouží ke statické kontrole kódu v jazyce ABAP a ke kontrole všech objektů z hlediska korektnosti, účinnosti, zabezpečení, spolehlivosti a statistických informací.

Code Inspector pomáhá vývojářům dodržovat standarty a zásady tím, že upozorní vývojáře, že jeho kód není zcela optimální. Funkce nabízí různé možnosti jak definovat soustavu objektů a také jak zkombinovat jednotlivé typy kontroly určitých prvků kódu do takzvaných variant kontroly. Tyto varianty poté umožňují komplexní kontrolu kódu nebo kontrolu více prvků a částí kódu najednou z různých úhlů pohledu.

Code Inspector je integrační platforma, která při spuštění přes transakci SCI umožňuje využití jednotlivých funkcí, které jsou zde dále také popsány, z jednoho místa. [7]

4.1.1 Scénáře použití

Code Inspector v jazyce ABAP má 3 různé scénáře pro použití, přičemž se u každého z ynich postupuje odlišným způsobem. Jedná se o možnost kontroly jednoho objektu, kontroly transportu a kontroly vybraných objektů. [7]

Kontrola jednoho objektu

Prvním scénářem je Develpment Workbench, který slouží pro kontrolu jednotlivých objektů. V závislosti na tom, co přesně potřebuje vývojář zkontrolovat, se liší použité vývojové prostředí a transakce, ve které se poté Code Inspector spustí.

- Object Navigator (transakce SE80)
- ABAP Editor (transakce SE38)
- Function Builder (transakce SE37)
- Class Builder (transakce SE24)
- ABAP Dictionary (transakce SE11)

V těchto jednotlivých vývojových prostředích se poté jednoduše spustí Code Inspector přes menu objekt – kontrola – Code Inspector. Objektem v tomto případě může být program, funkční modul, třída nebo tabulka. Jednotlivé objekty jsou potom kontrolovány pomocí aktuálně nastavené varianty kontroly. [7]

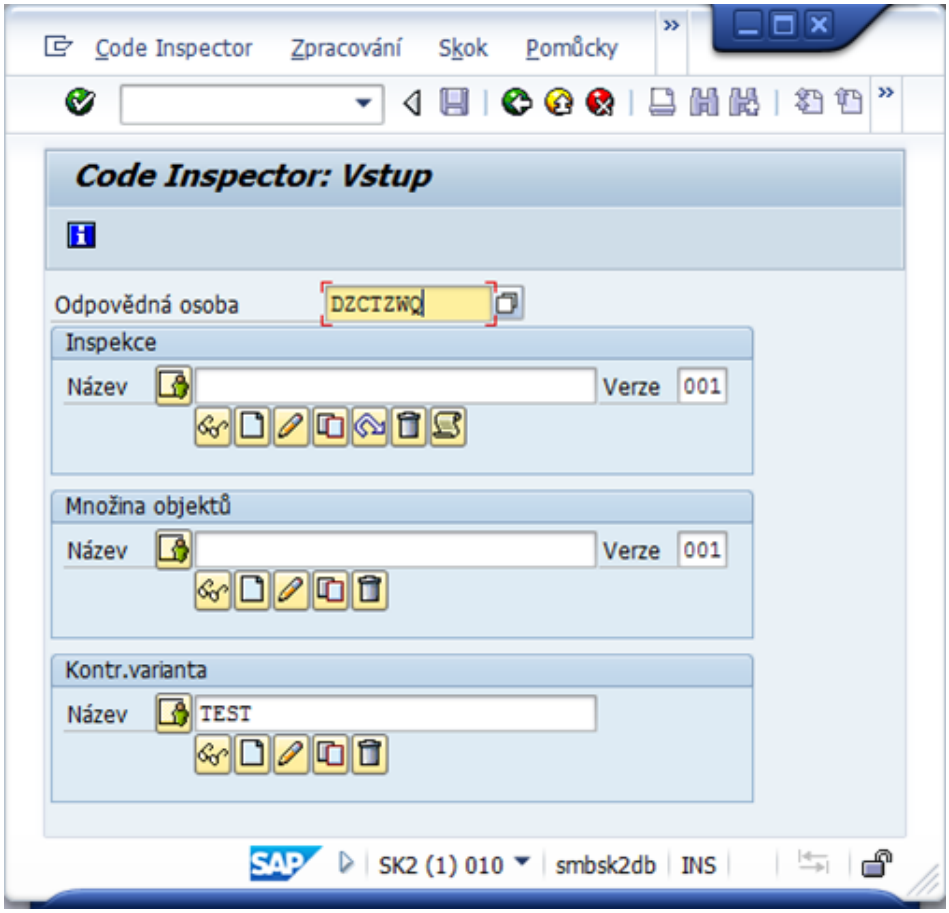
Kontrola transportu

Dalším scénářem je Transport Organizer, který umožňuje kontrolu všech objektů zahrnutých v daném transportním požadavku. Spouští se pomocí transakce SE09. Samotná kontrola se provádí označením požadovaného transportu a poté přes menu Požadavek/úloha - Celk. kontrola - Objekty (kontrola syntaxe). [7]

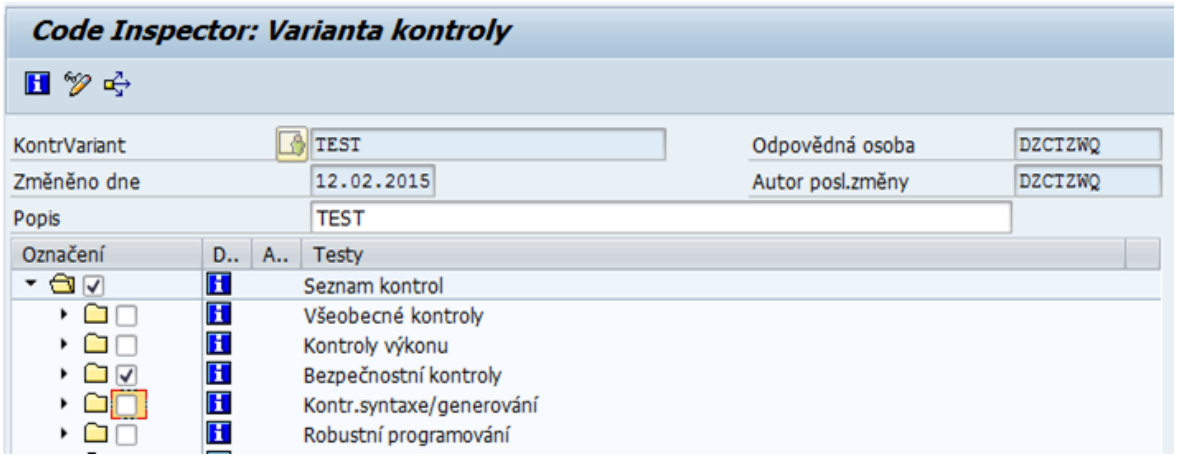
Kontrola vybraných objektů

Třetím scénářem je transakce SCI, která představuje samotný Code Inspector. Tato transakce umožňuje kontrolu celých skupin objektů dle vlastního výběru podle zadaných kritérií.

Těchto kritérií je celá řada, patří mezi ně například paket, softwarová a aplikační komponenta, zdrojový systém, transportní vrstva, zodpovědná osoba, typ objektu, název objektu a tak dále. Navíc také obsahuje speciální sběrače kolektorů, které umožnují například načíst objekt ze souboru. Skupiny objektů mohou být kombinovány s variantami kontroly k vytvoření takzvané inspekce, která může být spuštěna v jediném samostatném procesu. Na obrázku 1 lze vidět výstřižek z transakce SCI i s možnostmi výběru jednotlivých kritérií. Obrázek 2 ukazuje další možnosti výběru kritérií po zvolení varianty kontroly. [7]



Obrázek 5: Code inspector: Vstup
Zdroj: Prezentace ke školení z týmového webu oddělení EOA



Obrázek 6: Code Inspector: Varianta kontroly
Zdroj: Prezentace ke školení z týmového webu oddělení EOA

Code Inspector umožňuje kontrolu různých prvků programu, ty nejdůležitější z nich jsou vypsány níže. Dále je také možné vytvářet vlastní typy kontrol pro jiné prvky či kombinovat jednotlivé prvky do takzvaných kontrolních variant.

- Syntaxe
- Účinnost
- Bezpečnost
- Robustnost
- Programová konvence – názvy

Dále také nabízí pomocné funkce, které usnadňují práci s kódem.

- Vyhledávací funkce
- Metrika a statistika [7]

4.2 ABAP Test Cockpit

Poslední z těchto funkcí, tedy ABAP Test Cockpit je funkcí poměrně novou, do systému SAP byla implementována teprve v roce 2012, nicméně je to poněkud komplexnější funkce než ostatní jmenované, proto je v této práci popsána detailněji.

ABAP Test Cockpit je vlastně něco na způsob vylepšeného Code Inspectoru. Byl vyvinut za účelem umožnit vývojáři ještě důkladnější a propracovanější kontrolu kódu. ABAP Test Cockpit je novinka mezi způsoby kontroly kódu v jazyce ABAP, která umožňuje vývojáři provádět statickou kontrolu kódu a test jednotek pro programy. K zajištění hladkého průběhu kontroly a spolehlivých výsledků je tato funkce plně kompatibilní s funkcí Code Inspector. To znamená, že lze přenést vlastní nastavení pro varianty kontroly vytvořené v Code Inspectoru do ABAP Test Cockpitu.

ABAP Test Cockpit by měl být také v dohledné době integrován do Solution Manageru, integrované platformě, sloužící k podpoře a správě aplikací typu SAP i všech ostatních aplikací v rámci prostředí ERP. [8]

4.3 Transakce SLIN

Dalším pomocníkem vývojáře při kontrole kvality programovacího kódu je transakce SLIN, neboli ABAP Extended Program Check. Tato transakce umožňuje rozsáhlou kontrolu zvoleného programu. Uživatel má na výběr z velkého množství kategorií, které chce v rámci kódu zkontrolovat, k dispozici má zabudovanou nápovědu, která mu umožní rychle pochopit účel jednotlivých kategorií.

Výsledkem takovéto kontroly je potom tabulka obsahující přehled s výsledky. Tedy počty chyb, varování a zpráv pro jednotlivé kategorie. Z této tabulky se lze dostat pouhým dvojklikem na detailní popis nalezených problému v kódu. Odsud se lze snadno dostat k samotnému kódu na místo obsahující daný problém a dle potřeby ho i opravit. Po opravě je kontrolu doporučováno spustit znovu, aby mohl uživatel zjistit, zda problémy s kódem

setrvávají. Pokud ano, tento proces by měl být uživatelem opakován, dokud problémy nezmizí úplně.

Oproti Code Inspectoru a ABAP Test Cockpitu má ale i jisté nevýhody, nelze v něm kontrolovat více objektů najednou, jako jde například pomocí kontrolních variant v Code Inspectoru. Slouží pouze k detailní kontrole jednotlivých programů. Navíc zde není možnost zobrazit žádný souhrnný přehled o nalezených problémech z celé skupiny programů. Tato transakce je tedy vhodná spíše pro pravidelnou a komplexnější kontrolu jednoho programu. [9]

4.4 Code Profiler

Novinkou na poli kontroly kvality programovacího kódu v jazyce ABAP je Code Profiler. Tato funkce nepatří do základní výbavy systému SAP používaného ve společnostech, jedná se o nadstandartní funkci, která umožňuje automatickou kontrolu kvality programovacího kódu na vysoké úrovni. Na rozdíl od transakce SLIN nebo ABAP Test Cockpitu tato funkce není integrována v Code Inspectoru, nýbrž se jedná o samostatně pracující nástroj ke kontrole kódu.

V případě pořízení tohoto systému společností dochází k plné integraci do systému SAP. Po integraci již Code Profiler umožňuje přístup k celé řadě funkcí. Code Profiler je automatizovaný do větší míry než ostatní způsoby kontroly programovacího kódu. Využívá předem připravené informace z takzvaných testovacích případů použití, které pokrývají velkou většinu problémových situací, jež mohou nastat. To vše zajišťuje, že programy vytvářené pod dohledem Code Profileru by měly být oproštěny od běžně se vyskytujících chyb týkajících se bezpečnosti a kvality. Code Profiler tedy ochraňuje systém SAP proti neoprávněnému přístupu a zajišťuje splnění požadavků pro interní i externí audit. Navíc ještě může zvýšit výkon systému SAP a redukovat operační náklady.

Code Profiler je vlastně něco na způsob Firewallu pro jazyk ABAP, protože zabraňuje tomu, aby se žádný řádek nesprávného kódu nedostal z vývojového do produktivního systému

SAP. Toho je docíleno pomocí proaktivního přístupu Code Profileru, který automaticky pomáhá vývojáři upozornit na chyby a potenciální hrozby v kódu a následně je opravit. [10]

4.5 SAP Coverage Analyzer

Funkce SAP Coverage Analyzer (dále jen SCOV) slouží ke sběru a analýze takzvaného pokrytí kódu. Tento termín vyjadřuje, do jaké míry je daný program nebo modul otestován buď jednotlivými testovacími funkcemi, nebo ověřen dlouhodobým používáním v produktivním systému. Vysoký stupeň pokrytí kódu je známkou toho, že daný program byl dostatečně prověřen, tedy že většina jeho kódu již byla zatížena při používání.

K vyjádření této hodnoty slouží právě transakce SCOV a jí odpovídající funkce SAP Coverage Analyzer. Tato funkce vytvoří report s informacemi o pokrytí kódu z hlediska procedur, větvení a výrazů. Také sleduje četnost, s jakou je program spouštěn a množství výskytu runtime chyb, tedy softwarových chyb, které brní správnému průběhu programu. Možnosti využití této funkce se liší v závislosti na tom, zda ji používá vývojář či manažer kvality.

Vývojáři tato funkce slouží k určení míry, do jaké je jeho kód prověřen testy. Může efektivně najít mezery v testech a zaplnit je novými nebo modifikovanými testy. Takto může ověřit, zda integrační testy odpovídají z hlediska pokrytí kódu požadovaným hodnotám.

Manažeru kvality tato funkce pomůže sledovat hodnotu úrovně pokrytí kódu, získanou testováním ve QA systému (Quality Assurance System) a následně pak tyto údaje analyzovat a zpracovat je do reportu. Také ho může využít k vyhodnocení účinnosti jednotlivých systémů testujících kvalitu kódu. [11]

5 Kontrola kvality kódu ve Škoda auto

Nyní se již dostáváme k analytické části této práce, která se bude zabývat procesy kontroly kvality programovacího kódu v IT oddělení Škoda auto. S vývojem celé firmy, která stoupá stále vzhůru v důležitosti a množství spokojených zákazníků, se na vyšší úroveň dostává i práce IT týmu. Proto je kladen stále větší a větší důraz na kvalitu vyvíjených programů. S tím je neodmyslitelně spojená i kontrola kvality kódu, která slouží vývojáři i auditorovi kvality k ověření, zda je kód programu skutečně „kvalitní“.

Kvalita kódu v IT oddělení Škoda auto definuje podle množství kritérií. Vedle prvků technické kvality zmíněných v teoretické části práce kód v jazyce ABAP nesmí obsahovat celou řadu výrazů a naopak musí obsahovat určité formální prvky, jako je komentář úplně na začátku programu, ještě před samotným kódem. V neposlední řadě musí být název programu i jednotlivé proměnné v souladu se jmennou konvencí.

V předchozích letech v oddělení EOA nebyl brán příliš velký zřetel na kontrolu programovacího kódu, respektive nebyla nijak povinná a přikázána žádnou vyhláškou či pravidlem. Až v roce 2013 byla stanovena „Pravidla pro zajištění kvality zákaznického vývoje“ a s tím byla vydána i stejnojmenná příručka sloužící vývojářům.

5.1 Pravidla pro zajištění kvality zákaznického vývoje

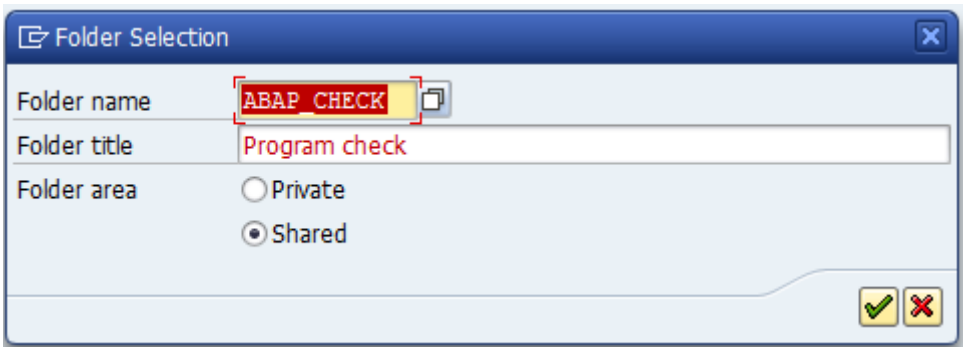
Tato příručka popisuje pravidla, která by měla pomoct zajistit kvalitu zákaznického vývoje. Tato pravidla jsou závazná a všichni vývojáři zodpovědní za vývoj v systému SAP pro společnost Škoda auto jsou povinni se dle ní řídit.

V příručce bylo stanoveno, že všichni vývojáři jsou od jejího uvedení v platnost povinni používat během vývoje programu funkci Code Inspector a ukládat jím předložené výsledky. Dále je v příručce výrazně doporučenou používání funkce SLIN (Extended syntax check). Nálezy se dělí do několika kategorií, některé z nich nepředstavují velkou míru rizika a tak mohou být dokonce ignorovány. Aktuální přehled jednotlivých kategorií nálezů je vždy

uveden na týmovém webu pro vývojáře. Pokud program obsahuje nálezy, které jsou v přehledu vedené jako závažné a nelze je tak ignorovat, a vývojář je přesto chce ponechat ve zdrojovém kódu (například z důvodu míry složitosti a časové ztráty v případě přepisování kódu), může tak udělat, za předpokladu, že o situaci informuje emailem oddělení Škoda CCoE (Koncernové centrum systému SAP – oddělení EOA/6) a vysvětlí své důvody. Tento email může vývojář vygenerovat jednoduše přímo z Code Inspectoru.

5.1.1 Protokol z Code Inspectoru

Po proběhnutí funkce Code Inspector jsou výsledky uloženy ve stromové formě. Vývojář má za úkol výsledky převést na formu list a uložit je ve sdíleném adresáři s názvem ABAP_CHECK. Vývojář však nezodpovídá za archivaci těchto logů a dále se jimi nezabývá, dokud není osloven auditory kontroly.



Obrázek 7: Archivace logů z Code Inspectoru
Zdroj: Výstřižek ze systému SAP

6 Automatická kontrola transportů v oddělení EOA

V roce 2014 poté přišla společnost VW Rusko, kterou podporuje oddělení EOA, s požadavkem na to, aby byl každý transport v systému SAP kontrolován. Vedení oddělení rozhodlo, že tato kontrola se bude provádět nejen pro VW Rusko, nýbrž i pro všechny ostatní externí společnosti, které oddělení EOA podporuje. To vedlo k zahájení vývoje programu, který by automaticky kontroloval chyby ve všech transportech a také ke stanovení rolí pro jednotlivé pracovníky oddělení EOA, kteří jsou součástí týmu zodpovídajícího za kontrolu kvality kódu. Každá z těchto rolí má nějakou funkci a s ní spojené povinnosti.

6.1 Role pro kvalitu vývoje zákaznických programů

6.1.1 Administrátor

Uživatel s touto rolí má za úkol přidělovat a spravovat oprávnění pro jednotlivé uživatele, které jsou součástí procesu k zajištění kvality vývoje. Dále má na starosti správu designu Týmového webu a aktivaci nepravidelně doplňovaných částí webu jako jsou odkazy, kontakty nebo emaily. Mezi jeho další povinnosti patří generování nepravidelně se opakujících akcí v systému, jako je prvotní nastavení statistik, aktualizace týmových web stránek nebo kontrola správnosti navrhovaných řešení pro odstranění chyb. V neposlední řadě také spravuje obsah nápovědy v systému.

6.1.2 Auditor

Jeho úkolem je provádět pravidelnou kontrolu programů a transportů v systému. Následně s nalezenými nesrovnalostmi v kódu oslovuje programátory a snaží se s nimi dojednat postup pro nalezení optimálního řešení. Případně problém eskaluje na jejich koordinátory. Jeho povinností je také pravidelně informovat o výsledcích kontrol programů a transportů management a koordinátory jednotlivých pododdělení. Poté, co je programátory

kontaktován o vyřešení jednotlivých nálezů, také provádí kontrolu těchto řešení. V případě, že je oddělení zadán nějaký projekt, se auditor podílí na stanovení vhodné strategie pro nastavení pravidel kvality při vývoji daného projektu.

6.1.3 Management

Management určuje strategii oprávnění jednotlivým uživatelům, kteří se účastní procesu zajištění kvality vývoje. Dále sleduje a zhodnocuje výsledky kontrol programů z hlediska systému a přijímá nezbytná opatření pro nápravu vyskytujících se problémů. Pokud dojde k eskalování problému na koordinátora, tak manažer tento případ detailně sleduje a přijme potřebná opatření pro nápravu. Také schvaluje všechna řešení pro odstranění chyb.

6.1.4 Koordinátor

Funkcí koordinátora při procesu zajištěný kvality vývoje je schvalování oprávnění pro jednotlivé uživatele, především co se týče vývojářů a externích konzultantů. Dále má také za úkol sledovat výsledky kontroly programů a transportů z hlediska svého pododdělení a jednotlivých vývojářů. Následně nalezené chyby projednává s vývojáři, kteří jsou za ně zodpovědní, a schvaluje jimi navržené postupy vedoucí k odstranění těchto chyb.

6.1.5 Development quality control board

Uskupení s názvem Development quality control board (Dále jen DQCB), složené z několika vybraných koordinátorů sleduje výsledky kontrol programů a transportů z hlediska chyb. Následně se věnuje návrhu systémového řešení, které by vedlo k odstranění těchto chyb.

6.1.6 Vývojář

Uživatel s touto rolí kromě samotného vývoje programů zodpovídá také za jejich kvalitu. Proto provádí kontrolu nového, případně i starého vývoje podle stanovených pravidel, tedy sám při vývoji používá Code Inspector. Také má za úkol pravidelně sledovat schválená řešení pro odstranění chyb a provádět nápravu zjištěných chyb. Další náplní jeho práce při procesu zajištění kvality zákaznického vývoje je konzultovat společně s auditorem a koordinátorem aktuální problémy spojené s vývojem.

6.1.7 Organizátor v procesu kvality

Poslední rolí v tomto schématu je organizátor v procesu kvality. Jeho úkolem je generovat pravidelné reporty, které následně předkládá managementu a koordinátorům. Dále pravidelně doplňuje týmový web, především data do kalendáře, nahrává statistiky, grafy a zápisy z jednání *DQCB*. Tato jednání také sám organizuje a tyto zápisy z nich pořizuje. Organizuje i setkání vývojářů, které je oddělené od jednání DQCB. Na starosti má také organizaci školení, které jsou určeny nejen pro vývojáře, ale především pak pro začátečníky v procesu zajištění kvality vývoje zákaznických programů, tedy nováčky na oddělení EOA.

7 Automatická kontrola transportů

Nyní se již dostáváme k samotnému procesu automatické kontroly transportů, která je prováděna pomocí programu vyvinutého oddělením EOA. Jedná se o složitý a velmi propracovaný program, při jehož vývoji i následném zdokonalování je kladen velký důraz na to, aby byla kontrola co možná nejvíce automatická při zachování stejné míry efektivity kontroly. Tento program se stále ještě vyvíjí a zdokonaluje a s ním se vyvíjí také celý proces kontroly.

7.1 Rozsah a frekvence kontroly

Dle stávajících požadavků dochází ke kontrole všech programů, které se nacházejí v nějakém z uvolněných transportů v systémech SK a SE. Do budoucna je plánováno rozšíření na další zákaznické objekty a systémy. Kontrola probíhá každý pracovní den pomocí jobu s názvem *999_CC_CDMC_COLLECT_RESULTS,* který se spouští v klientu *SSM/999*. Kontrola se však týká všech klientů vývojových systému, pokud je alespoň jeden z nich nedostupný, job je ukončen a kontrola neproběhne.

7.2 Proces Kontroly

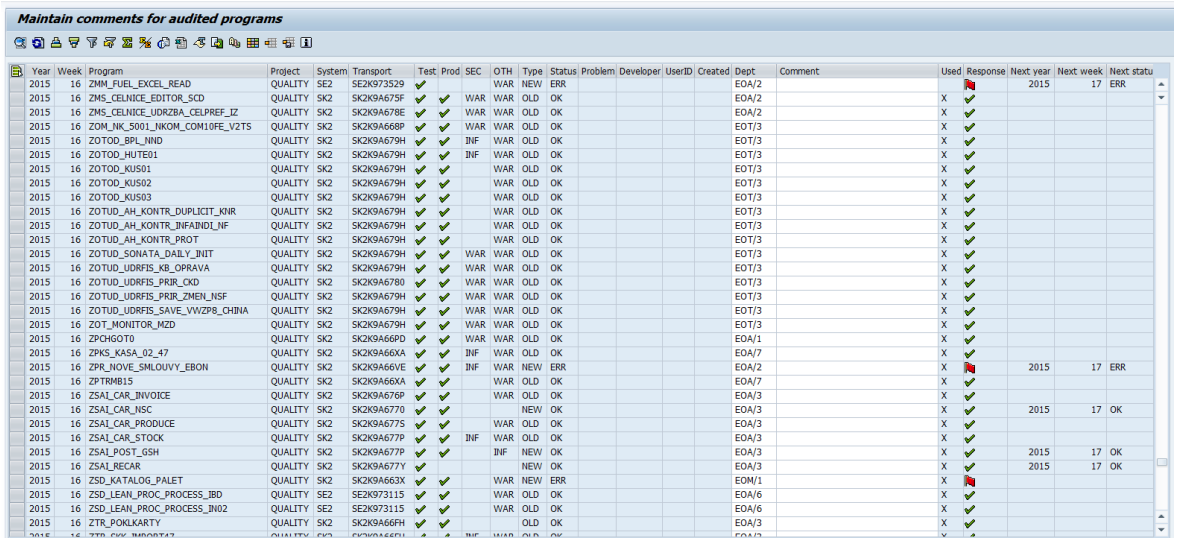
Nyní se již dostáváme k samotnému procesu kontroly programů, který si detailně popíšeme a vysvětlíme. Na úplném začátku je job, který posbírá informace o programech v uvolněných transportech za uplynulý den. Tento job musí být manuálně spouštěn, za což je zodpovědný pracovník v roli auditora. Po nasbírání těchto informací, respektive po nalezení chyb v kódech programů, obsažených v kontrolovaných transportech, vezme auditor tyto chyby a rozešle jejich seznam zodpovědným osobám, tedy koordinátorům jednotlivých oddělení ve formě sešitu Microsoft Excel. Tento proces se děje pouze jednou týdně.

Transakce pro kontrolu nálezů

Seznam kontrolovaných programů je pro auditora je dostupný ve výstupu transakce ZCC_CDMC, do níž má přístup pouze sám auditor. Výstup této transakce je ve formě tabulky s údaji o kontrolovaných programech. Auditor posbírá programy z tabulky a vloží je do seznamu v Excelu, který následně rozešle zodpovědným osobám, tedy jednotlivým koordinátorům.

7.2.1 Přiřazení programů k jejich autorům

Koordinátoři následně se svými kolegy tento seznam proberou, aby ověřili, zda byly kontrolované programy správně přiřazeny jejich oddělení. Pokud tyto programy nebyly vytvořeny nikým z jeho oddělení, kontaktuje koordinátor auditora, který se dál bude snažit nalézt autora daného programu. Poté, co auditor kontroly úspěšně nalezne autora, přidá jeho jméno a oddělení, na které patří, do seznamu pro příští týden. Tento postup se v případě potřeby opakuje do doby, než bude k danému programu přiřazen skutečný autor a oddělení, na kterém působí. Na obrázku č. 7 můžeme vidět výstup transakce ZCC_CDMC, tedy seznam kontrolovaných programů a informace o nich.



Year	Week	Program	Project	System	Transport	Test	Prod	SEC	OTH	Type	Status	Problem	Developer	UserID	Created	Dept	Comment	Used	Response	Next year	Next week	Next statu
2015	16	ZMM_FUEL_EXCEL_READ	QUALITY	SE2	SE2K973529	✔			WAR	NEW	ERR					EOA/2				2015	17	ERR
2015	16	ZMS_CELNICE_EDITOR_SCD	QUALITY	SK2	SK2K9A675F	✔	✔	WAR	WAR	OLD	OK					EOA/2		X	✔			
2015	16	ZMS_CELNICE_UDRZBA_CELPREF_IZ	QUALITY	SK2	SK2K9A678E	✔	✔	WAR	WAR	OLD	OK					EOA/2		X	✔			
2015	16	ZOM_NK_5001_NKOM_COM10FE_V2TS	QUALITY	SK2	SK2K9A668P	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTOD_BPL_NND	QUALITY	SK2	SK2K9A679H	✔	✔	INF	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTOD_HUTE01	QUALITY	SK2	SK2K9A679H	✔	✔	INF	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTOD_KUS01	QUALITY	SK2	SK2K9A679H	✔	✔		WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTOD_KUS02	QUALITY	SK2	SK2K9A679H	✔	✔		WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTOD_KUS03	QUALITY	SK2	SK2K9A679H	✔	✔		WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_AH_KONTR_DUPLICIT_KNR	QUALITY	SK2	SK2K9A679H	✔	✔		WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_AH_KONTR_INFAINDI_NF	QUALITY	SK2	SK2K9A679H	✔	✔		WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_AH_KONTR_PROT	QUALITY	SK2	SK2K9A679H	✔	✔		WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_SONATA_DAILY_INIT	QUALITY	SK2	SK2K9A679H	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_UDRFIS_KB_OPRAVA	QUALITY	SK2	SK2K9A679H	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_UDRFIS_PRIR_CKD	QUALITY	SK2	SK2K9A6780	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_UDRFIS_PRIR_ZMEN_NSF	QUALITY	SK2	SK2K9A679H	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOTUD_UDRFIS_SAVE_VWZP8_CHINA	QUALITY	SK2	SK2K9A679H	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZOT_MONITOR_MZD	QUALITY	SK2	SK2K9A679H	✔	✔	WAR	WAR	OLD	OK					EOT/3		X	✔			
2015	16	ZPCHGOT0	QUALITY	SK2	SK2K9A66PD	✔	✔	WAR	WAR	OLD	OK					EOA/1		X	✔			
2015	16	ZPKS_KASA_02_47	QUALITY	SK2	SK2K9A66XA	✔	✔	INF	WAR	OLD	OK					EOA/7		X	✔			
2015	16	ZPR_NOVE_SMLOUVY_EBON	QUALITY	SK2	SK2K9A66VE	✔	✔	INF	WAR	NEW	ERR					EOA/2		X		2015	17	ERR
2015	16	ZPTRMB15	QUALITY	SK2	SK2K9A66XA	✔	✔		WAR	OLD	OK					EOA/7		X	✔			
2015	16	ZSAI_CAR_INVOICE	QUALITY	SK2	SK2K9A676P	✔	✔		WAR	OLD	OK					EOA/3		X	✔			
2015	16	ZSAI_CAR_NSC	QUALITY	SK2	SK2K9A6770	✔	✔			NEW	OK					EOA/3		X	✔	2015	17	OK
2015	16	ZSAI_CAR_PRODUCE	QUALITY	SK2	SK2K9A677S	✔	✔		WAR	OLD	OK					EOA/3		X	✔			
2015	16	ZSAI_CAR_STOCK	QUALITY	SK2	SK2K9A677P	✔	✔	INF	WAR	OLD	OK					EOA/3		X	✔			
2015	16	ZSAI_POST_GSH	QUALITY	SK2	SK2K9A677P	✔	✔		INF	NEW	OK					EOA/3		X	✔	2015	17	OK
2015	16	ZSAI_RECAR	QUALITY	SK2	SK2K9A677Y	✔				NEW	OK					EOA/3		X	✔	2015	17	OK
2015	16	ZSD_KATALOG_PALET	QUALITY	SK2	SK2K9A663X	✔	✔		WAR	NEW	ERR					EOM/1		X				
2015	16	ZSD_LEAN_PROC_PROCESS_IBD	QUALITY	SE2	SE2K973115	✔	✔		WAR	OLD	OK					EOA/6		X	✔			
2015	16	ZSD_LEAN_PROC_PROCESS_IN02	QUALITY	SE2	SE2K973115	✔	✔		WAR	OLD	OK					EOA/6		X	✔			
2015	16	ZTR_POKLKARTY	QUALITY	SK2	SK2K9A66FH	✔	✔			OLD	OK					EOA/3		X	✔			

Obrázek 8: Seznam kontrolovaných programů
Zdroj: Výstřižek z výstupu transakce ZCC_CDMC

V tabulce č. 1 jsou vypsány nadpisy sloupců z výstupu této transakce a jejich popis s vysvětlením.

Tabulka 1: Výstup z transkace ZCC_CDMC

Year	Rok, ve kterém byl program kontrolován
Week	Týden, ve kterém byl program kontrolován
Program	Název kontrolovaného programu
Project	Druh kontroly (většinou z hlediska kvality)
System	Klient systému, ve kterém se kontrolovaný program nachází
Transport	Číslo transportu, ve kterém byl kontrolovaný program transportován
Test	Status říkající, zda transportní požadavek dorazil do testovacího systému bez chyby
Prod	Status říkající, zda transportní požadavek dorazil do produktivního systému bez chyby
SEC	Informace o tom, zda kontrolovaný program obsahuje security isssue
OTH	Informace o tom, zda kontrolovaný program obsahuje other issue
Type	Zařazení programu do kategorie NEW/OLD
Status	Definuje, jestli je potřeba k danému programu přidat komentář. (ERR – je třeba, OK – není třeba)
Problem	Popis vyskytujícího se problému

Developer	Jméno vývojáře, který program vytvořil nebo naposledy upravil.
UserID	Identifikační číslo vývojáře, který program vytvořil nebo naposledy upravil
Created	Datum vytvoření programu
Dept	Oddělení, které za daný program odpovídá
Comment	Komentář, který je třeba doplnit do programu s chybou
Used	Určuje, zda je kontrolovaný program používán v produktivním systému
Response	Status určující, zda je požadováno vyjádření zodpovědné osoby za daný program
Next year	Rok, ve kterém byl program kontrolován naposledy
Next week	Týden, ve kterém byl program kontrolován naposledy
Next status	Status programu při poslední provedené kontrole (ERR/OK)
Improved	Určuje, zda se změnil status programu od poslední kontroly (Kvalita kódu)

Zdroj: Vlastní zpracování

7.2.2 Klasifikace programů a nálezů Code Inspectoru

Programy podrobené procesu kontroly kvality kódu se ve Škoda auto dělí do dvou kategorií na staré a nové, v závislosti na tom, kdy byl který program vytvořen.

Staré programy

Do této kategorie patří všechny programy, které vznikly před 1. 12. 2014. Tyto programy vyloženě nesmí obsahovat pouze takzvanou *Bezpečnostní chybu*, i pro tento typ chyby ovšem existují výjimky. Pokud by předělání programu, který obsahuje tuto chybu, bylo finančně příliš nákladné, či by jeho dočasná nefunkčnost způsobila společnosti velké finanční ztráty, je po domluvě možné nechat daný program ve stávajícím stavu, tedy neopravený.

Nové programy

Programy z této kategorie vznikly po 1. 12. 2014. Tyto programy by neměly obsahovat jakýkoliv nález vykazující chybu, mohou obsahovat pouze informační hlášení, která nejsou z hlediska bezpečnosti tak riziková.

O tom, do jaké míry je situace ohledně kontrolovaného programu kritická a jak je s ní potřeba naložit, se rozhoduje podle typu nalezených chyb, ty se dělí do několika kategorií klasifikovaných podle Code Inspectoru, tyto kategorie jsou popsány v tabulce č. 2. Typy chyb se zeleným pozadím buňky jsou v pořádku a není třeba je ihned začít opravovat, typy s červeným pozadím buňky je naopak třeba neprodleně začít řešit a hledat cestu k jejich opravě.

Tabulka 2: Dělení nalezených chyb

	Bezpečnostní chyba	Jiná chyba
Nový program	Informační hlášení	Informační hlášení
	Varování	Varování
	Chybová hláška	Chybová hláška
Starý program	Informační hlášení	Informační hlášení
	Varování	Varování
	Chybová hláška	Chybová hláška

Zdroj: Vlastní zpracování

Jak lze vyčíst z tabulky, Bezpečnostní chyby, tedy chyby v kódu ohrožující bezpečnost celé společnosti Škoda auto jsou nejzávažnějším typem vyskytujících se chyb. Tyto chyby se vyskytují v případě, že se v kódu kontrolovaného programu nachází některý z takzvaných kritických příkazů. Tyto příkazy se u nově vyvíjených programů již nepoužívají, mohou se však nacházet v kódech starších programů. Patří mezi ně následující příkazy:

- CALL EDITOR.
- GENERATE &1.
- WRITE / INSERT / DELETE REPORT/ TEXT POOL.
- WRITE / READ SCREEN.
- WRITE NAMETAB.

Tyto příkazy jsou rizikové z několika důvodů. Tím hlavním je však skutečnost, že s těmito příkazy v kódu existuje riziko zneužití systému. Některý z uživatelů pracujících v systému SAP by mohl například změnit cílový účet, na který posílá platbu a místo toho jí odeslat na svůj vlastní.

7.2.3 Práce s nalezenými chybami

Programy, ve kterých job nenalezl žádnou chybu, není dále potřeba nijak zvlášť řešit. Pokud program chybu obsahuje a tato chyba spadá do kategorií označených červeně v tabulce č. 2, bude program podroben procesu sestávajícím se z kroků, které mají za cíl tyto chyby odstranit nebo alespoň odůvodnit.

Další úkol v procesu s cílem odstranění chyb v kódu programu připadá na vývojáře. Vývojář by měl k programu připojit komentář do seznamu programů v Excelu, který mu přepošle koordinátor.

Poté co všichni vývojáři připojí komentáře k požadovaným programům, koordinátor jednotlivé komentáře posbírá do seznamu a pošle jej zpět auditorovy kontroly. Ten je poté přidá do sešitu Microsoft Excel s názvem „Programy s chybou," který obsahuje seznam jednotlivých programů, ve kterých byla nalezena chyba. Auditor zde pak jednotlivé programy klasifikuje buď jako nový, nebo jako starý typ. Další postup se liší v závislosti na typu kontrolovaného programu.

Starý typ programu

U programu starého typu dochází k analýze komentářů, které do seznamu kontrolovaných programů v sešitu Microsoft Excel doplnila zodpovědná osoba za daný program, tedy vývojář. Na základě výsledků této analýzy se pracovníci zastávající jednotlivé role dohadují a snaží se co nejvhodněji rozhodnout, zda je program nutné opravit či nikoliv. Toto rozhodování se provádí na základě vyhodnocení nalezených chyb, tak jak jsou uvedeny v tabulce č. 2.

Pokud je program napsán způsobem, při kterém by oprava kódu byla příliš složitá a trvala příliš dlouho, rozhodne tým zodpovědný za kvalitu zákaznického kódu o tom, že kód může zůstat stejný. Případně se jeho oprava odloží na období, při kterém nebude vývojář daného programu příliš vytížen a bude mít tak čas věnovat se tak jeho opravě. Na základě tohoto rozhodnutí je programu v seznamu *Programy s chybou* nastaven příznak *Nebude se opravovat,* tedy v případě že je rozhodnuto pro zanechání programu ve stávající formě. Pokud je rozhodnuto o opravě daného programu, postupuje se u něj úplně stejně jako v případě programů nového typu, které musejí být opraveny bezpodmínečně všechny.

Nový typ programu

V případě programu nového typu je postup poněkud jiný, platí však i pro programy starého typu, u kterých se rozhodne pro jejich opravu. U těchto programů se nejprve zhodnotí náročnost opravy a následně je na základě toho stanoveno datum, před kterým by měl být program opraven. Ne vždy je však toto datum možné dodržet, zejména z důvodu vytíženosti pracovníka zodpovědného za daný program. Když nastane den, do kterého měl být program opraven, zkontroluje auditor stav programu, pokud stále opraven není nebo je opraven nedostatečně, obrací se auditor zpět na vývojáře a požaduje po něm vysvětlení. Následně se společně snaží dohodnout na novém datu, do kterého by měl vývojář být schopný uvést program do požadovaného stavu. Když je program konečně opraven, je třeba ho ještě odstranit ze seznamu *Programy s chybou* a uvědomit o tom zbytek týmu zajišťujícího kvalitu zákaznického vývoje v oddělení EOA ve Škoda auto.

7.2.4 Zhodnocení vývoje procesu kontroly kódu

Takto lze definovat a přiblížit postup při procesu kontroly kvality programovacího kódu ve Škoda auto pro čtenáře této práce, neznalého prostředí oddělení EOA z vlastní zkušenosti.

Tento proces ve své stávající podobě ještě stále není úplný. Tým zodpovědný za kvalitu vývoje v oddělení EOA se stále snaží nalézt způsoby, jak tento proces co nejvíce zefektivnit. S tím je samozřejmě spojena i úprava kódu programu, který tuto kontrolu provádí. Tyto nápady a úpravy je složitější převést do podoby programovacího kódu, než pouze vymyslet jako funkce, tento fakt vývoj procesu značně ztěžuje. Nicméně o kód samotný se stará vývojář, který má více než dvaceti leté zkušenosti s vývojem v jazyce ABAP, takže je většinu nápadů na zlepšení možné realizovat.

Proces kontroly kódu, který se zde používá, je ve srovnání s ostatními přístupy ke kontrole velmi efektivní a také daleko více automatický, přesto má však stále určité nedostatky, které budou popsány v další kapitole. Kromě popisu těchto nedostatků zde budou také návrhy, které by mohly pomoct proces kontroly kvality optimalizovat a ještě více zautomatizovat.

8 Návrhy ke zlepšení procesu kontroly zákaznického kódu

Jak již bylo zmíněno o kapitolu výše, proces automatické kontroly kvality kódu v oddělení EOA společnosti Škoda auto má jisté nedostatky. Ty budou nastíněny v této kapitole a společně s nimi zde budou prezentována možná řešení těchto problémů, která jsem vymyslel ve spolupráci s týmem odpovědným za kontrolu kvality kódu.

8.1 Zpětná kontrola transportů

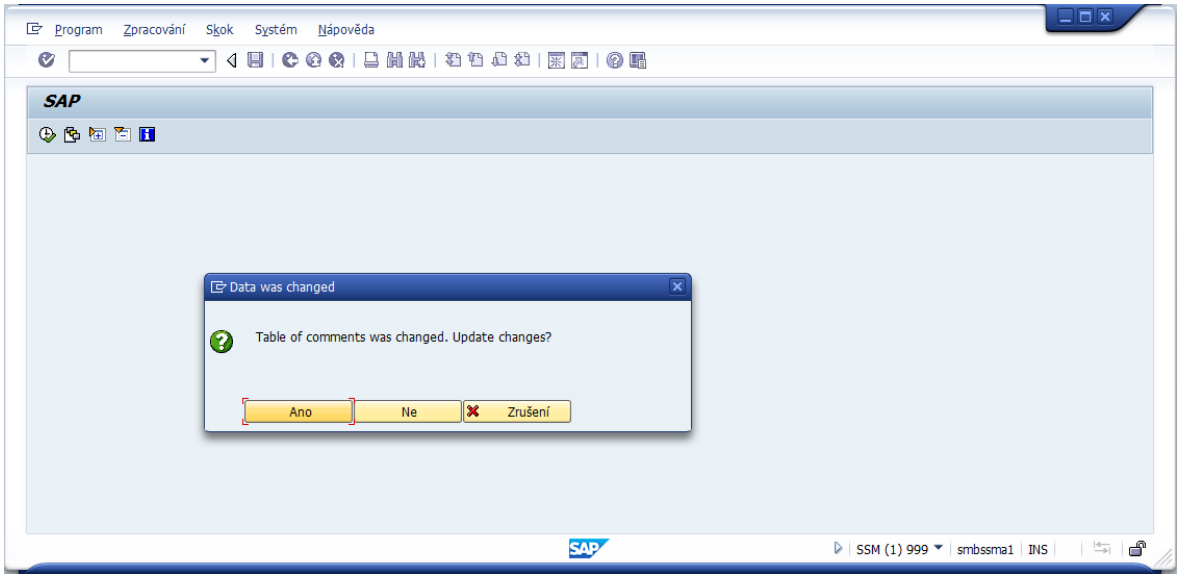
Jedním z problémů týkajících se procesu kontroly je fakt, že kontrola transportů probíhá zpětně. To znamená, že i když kontrolní program najde v kontrolovaných transportech chyby v kódu, transport je přesto uvolněn. To se děje zejména z toho důvodu, že některé transporty nelze zadržet i přesto, že by mohly představovat riziko pro společnost Škoda auto. Zadržení těchto transportů by mohlo například způsobit, že některé z oddělení by nemohlo efektivně vykonávat svoji práci, což by mohlo vést až k ušlým ziskům pro společnost Škoda auto.

Nalézt úplné řešení pro tento problém není vůbec snadné, pravděpodobně ani dost dobře možné. Jednou z možností by bylo rozlišovat důležitost kontrolovaných transportů. Tedy například u těch méně důležitých transportů z hlediska fungování společnosti, by kontrola neprobíhala zpětně, ale přímo by zamezovala projití transportu do produkce. Nicméně by bylo velice těžké stanovit kritéria pro rozlišení důležitosti jednotlivých transportů a programů s nimi spojených. Stejně tak by bylo těžké tuto změnu implementovat do kódu programu, který kontroluje transporty.

8.2 Pop-up okno u transakce ZCC_ CDMC

Další z věcí, kterou sice nelze považovat za velký problém, nicméně do jisté míry snižuje efektivitu procesu kontroly z hlediska auditora, je pop-up okno u transakce ZCC_CDMC.

Toto okno se objeví pokaždé, když chce auditor uložit změny ve výstupu transakce ZCC_CDMC, kam k jednotlivým programům přidává komentáře, které nasbíral koordinátor od vývojářů a přidal je do sešitu Microsoft Excel s názvem *Programy s chybou*, který následně odeslal auditorovi zpět. Tato skutečnost vyžaduje při každém uložení změny další zbytečné kliknutí navíc, kterých je v systému SAP již tak dost.

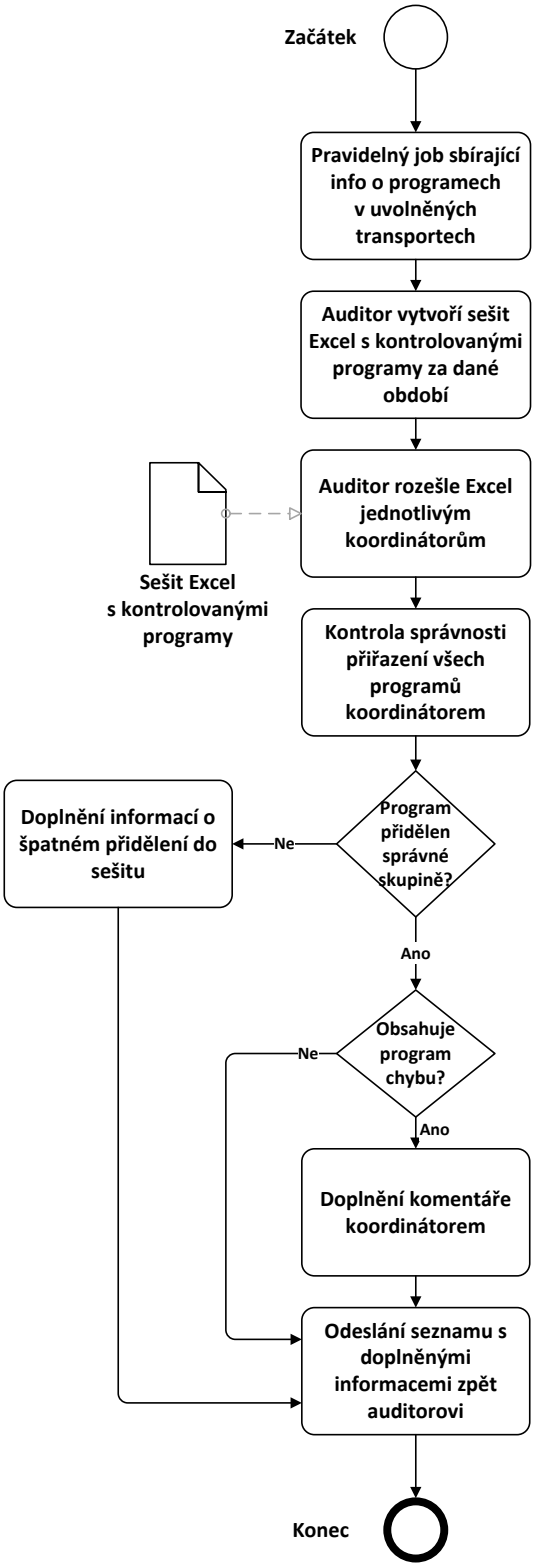


Obrázek 9 : Pop-up okno při ukládání změn v transakci ZCC_CDMC
Zdroj: Výstřižek z transakce ZCC_CDMC

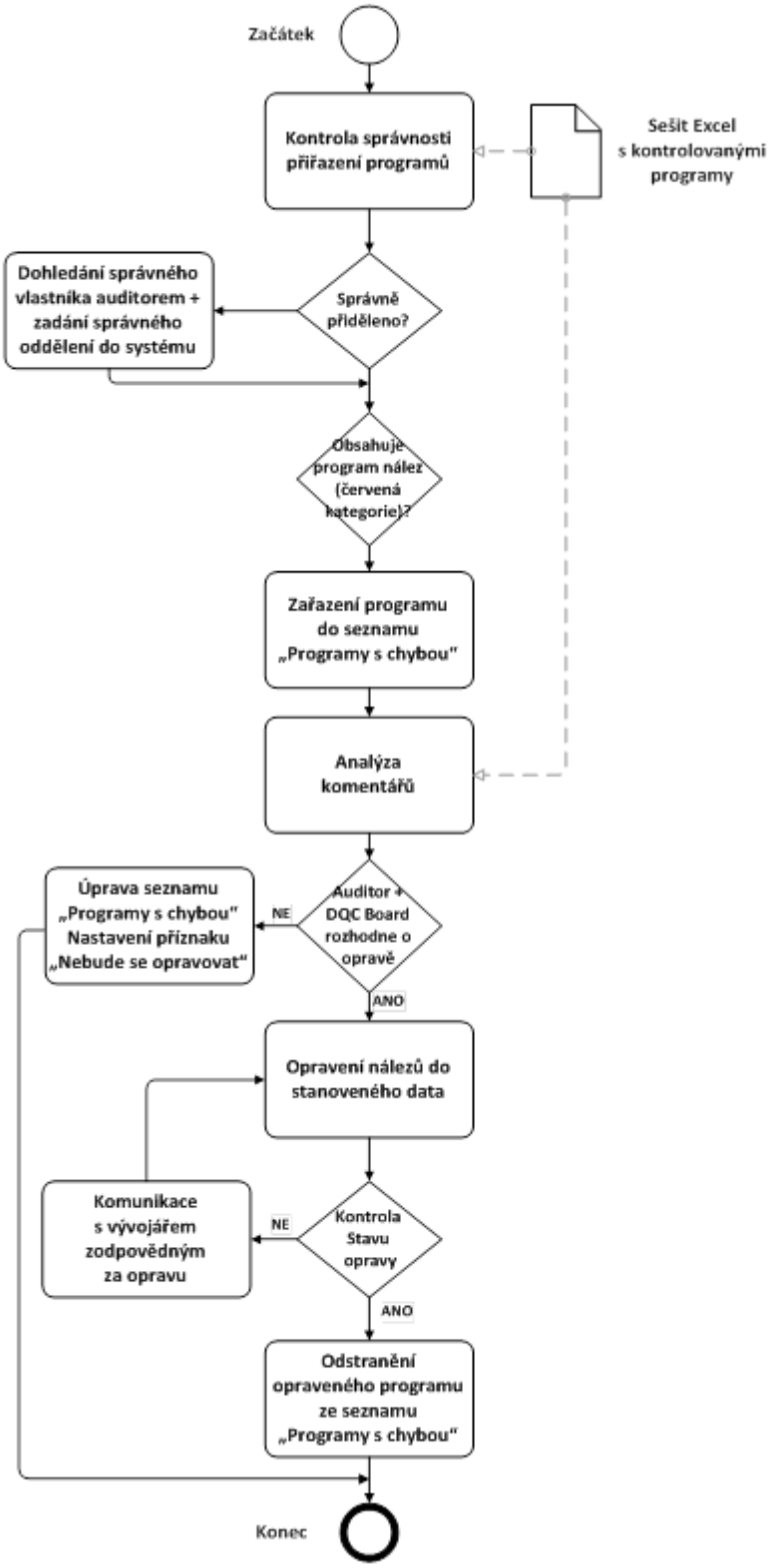
Řešení tohoto problému je vcelku jednoduché, z hlediska procesu kontroly nemá tato změna téměř žádný vliv a z hlediska nutnosti upravit zdrojový kód programu, který provádí kontrolu transportů, se také nejedná o příliš náročnou operaci. Více než k čemukoliv jinému by tato změna sloužila ke zjednodušení práce pro auditora kontroly.

8.3 Stanovení nové procedury pro kontrolu programů

Posledním a zároveň nejzásadnějším nápadem na zlepšení kontroly kvality kódu je návrh nové, upravené procedury automatické kontroly programů. Procedura ve svém současně používaném stavu je rozdělena do dvou kroků, vyjádřených v diagramech na obrázcích č. 10 a 11.



Obrázek 10: Stávající postup při získávání odpovědí od koordinátorů
Zdroj: Vlastní zpracování



Obrázek 11: Stávající postup při vyhodnocování a opravě chyb
Zdroj: Vlastní zpracování

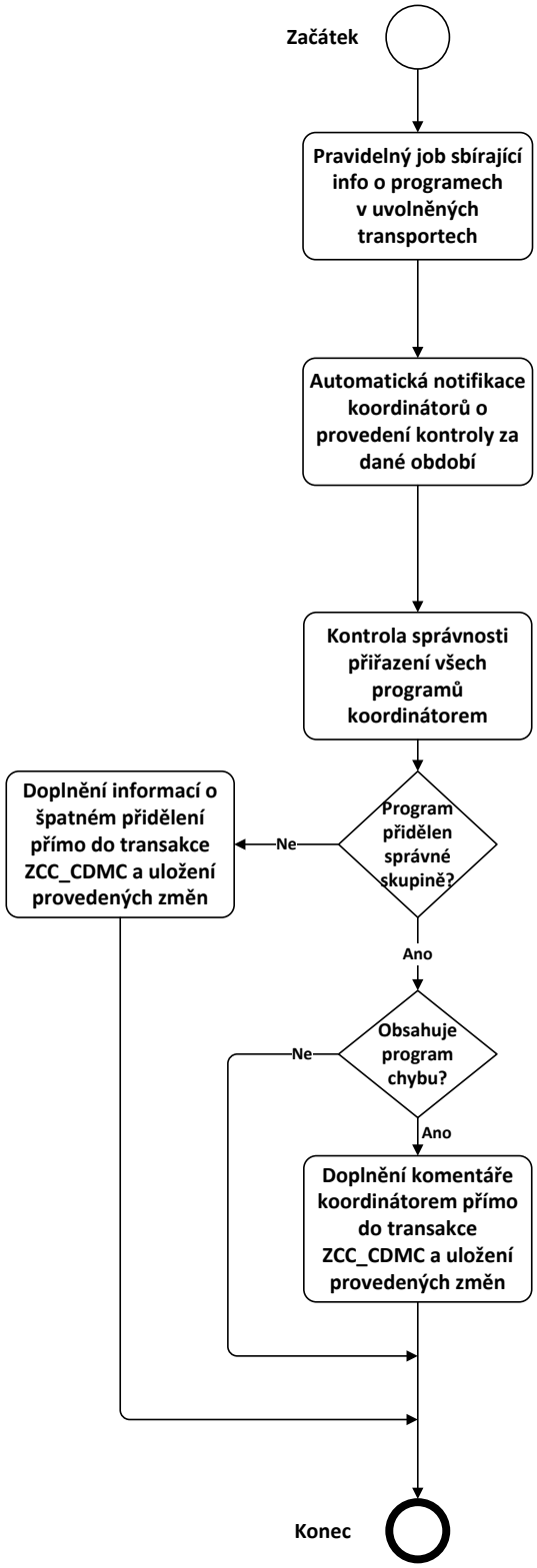
Jak lze vidět s výše uvedených obrázků, procedura je rozdělena na 2 kroky, přičemž nově navržené řešení se týká především prvního kroku procedury. Hlavní nedostatek při prvním kroku je složitost získávání komentářů od vývojářů skrze koordinátora. Auditor musí data získaná z výstupu transakce ZCC_CDMC zpracovat do seznamu v sešitu Excel a následně jej rozeslat všem koordinátorům, kteří dále oslovují vývojáře spadající pod jejich oddělení, aby mohli komentář k programu přidat do seznamu. Tento seznam poté opět posílají zpět auditorovi, který musí všechny seznamy spojit do jednoho, což je velmi zdlouhavé a nepraktické.

Tento problém by se mohl vyřešit přidáním autorizačních objektů do kódu transakce ZCC_CDMC. Pomocí autorizačních objektů by se pro jednotlivé uživatele této transakce daly vytvořit role, ke kterým by byla přidělena oprávnění. Po vytvoření těchto rolí by transakci ZCC_CDMC již nepoužívali pouze auditoři, nýbrž také koordinátoři a vývojáři. Vzhled a dostupné funkce této transakce by se poté lišily na základě role uživatele, který jí používá.

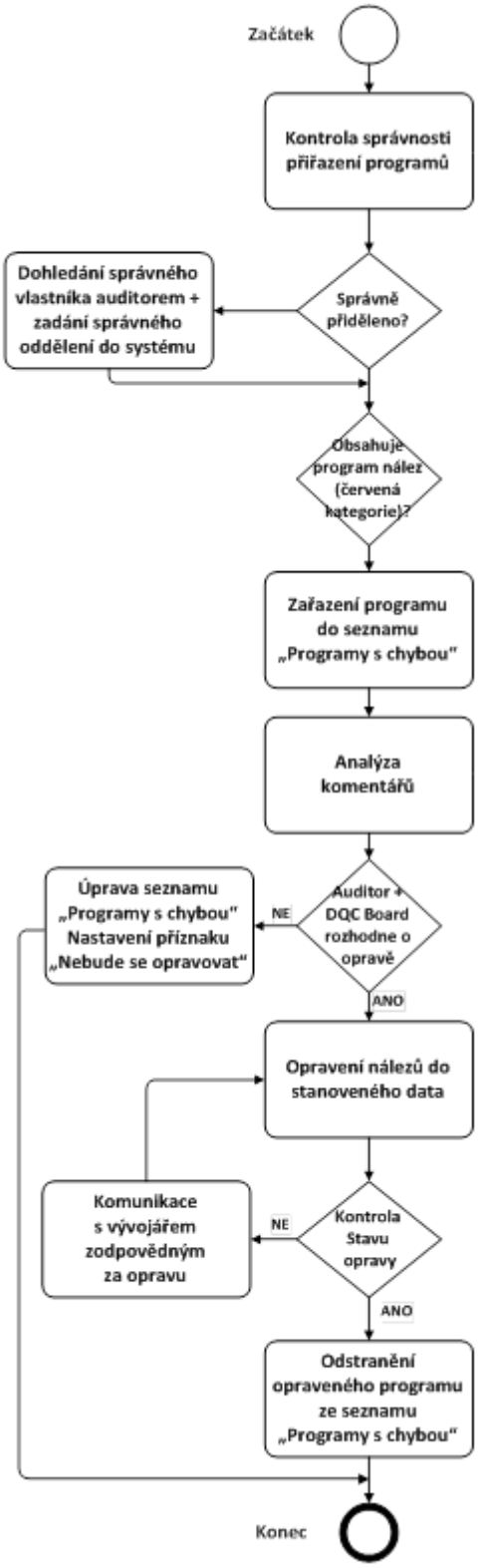
Hlavní výhodou této změny by byl konec povinnosti auditora zpracovávat výstup této transakce do seznamu v sešitu Excel. Každý by měl do této transakce přístup a viděl v ní pouze ty programy, které se ho týkají. Koordinátor by tak mohl snadno upozornit jednotlivé vývojáře na chyby v programech, za které jsou zodpovědní a vývojáři by mohli komentáře jednoduše přidat osobně přímo v transakci ZCC_CDMC.

Druhý krok procedury, tedy vyhodnocování a oprava chyb by příliš ovlivněn nebyl, protože se při něm tolik nepracuje s transakcí ZCC_CDMC. Jediná změna v tomto kroku by se týkala kontroly správnosti přiřazení programů, kdy by auditor místo seznamu s programu v sešitě Excel zkontroloval správnost přímo v transakci ZCC_CDMC. Zbytek postupu v druhém kroku procedury by zůstal nezměněn.

Na obrázcích č. 12 a 13 můžeme vidět diagramy obou kroků nově navržené procedury.



Obrázek 12: Nově navržený postup při získávání odpovědí od koordinátorů
Zdroj: Vlastní zpracování



Obrázek 13: Nově navržený postup při vyhodnocování a opravě chyb
Zdroj: Vlastní zpracování

8.4 Zhodnocení nově navržené procedury

Nově navrženou proceduru zatím není zcela možné prověřit používáním v praxi, protože ještě nebyla zcela vytvořena ani implementována. Nicméně podle zbytku týmu zajišťujícího kontrolu kvality zákaznického kódu je tento návrh vcelku realizovatelný. Tým zastává ten názor, že nově navržená procedura by měla pomoct zefektivnit proces kontroly kvality kódu v oddělení EOA. Především by pak do značné míry zvýšila úroveň automatizace tohoto procesu, jelikož by při zavedení této procedury značně ubylo potřeby komunikace mezi jednotlivými pracovníky zodpovědnými za kontrolu kvality kódu. Úroveň přínosu tohoto návrhu mohou ukázat až následující měsíce, podle toho, zda se tým zodpovědný za kvalitu kódu v oddělení EOA rozhodne při kontrole začít postupovat podle nově navržené procedury.

Závěr

Cílem této bakalářské práce bylo přiblížit používané přístupy ke kontrole kvality programovacího kódu, popsat nový postup, který je aktuálně stále ještě ve vývoji v oddělení EOA společnosti Škoda auto. Dále bylo také cílem tento vývoj zhodnotit a na základě tohoto hodnocení navrhnout změny, které by ho pomohly zefektivnit. Posledním úkolem bylo pokusit se objektivně zhodnotit navržené změny k procesu kontroly. To vše na základě získaných zkušeností ze spolupráce s interními týmy vývojářů a ostatních pracovníků figurujících v procesu zajištění zákaznické kvality kódu v oddělení EOA.

Základem pro vypracování bakalářské práce byla teoretická část, která vychází z informací načerpaných z odborné literatury věnující se tématům jako je systém SAP, základy jazyka ABAP a zdokonalování vývoje v jazyce ABAP. Teoretická část představuje prvky, podle kterých se kvalita programovacího kódu definuje, dále detailně popisuje systém SAP a jazyk ABAP, který je v něm integrován. Nakonec představuje dostupné a všeobecně používané metody kontroly programovacího kódu v jazyce ABAP.

Praktická část byla zaměřena na detailní popis nově vyvíjené metody kontroly zákaznického kódu v oddělení EOA společnosti Škoda auto, která je již během svého vývoje v provozu. Proces kontroly při použití této metody byl v této práci důkladně popsán.

Nejprve byla v práci popsána stručná historie a důvod pro vznik vývoje této metody. Dále zde byly také charakterizovány jednotlivé role pracovníků účastnících se procesu kontroly kódu v oddělení EOA. Další kapitola praktické části se zabývala popisem samotného procesu kontroly kódu, tedy jak často se provádí, jaký je postup a jakou úlohu v něm mají jednotliví účastníci.

Poslední kapitola se věnuje návrhům, které by mohly pomoct zdokonalit a ještě více zautomatizovat tento proces. To byl také jeden z hlavních cílů pro tuto práci, nalézt v procesu kontroly nedostatky, které by šly alespoň částečně odstranit. Zejména je zde popsán návrh nové procedury pro automatickou kontrolu kvality programovacího kódu v oddělení EOA.

Do jaké míry je tento i ostatní návrhy na zlepšení přínosný, se teprve ukáže v následujících měsících, až bude mít tým zajišťující kontrolu kvality kódu čas na zvážení těchto návrhů a jejich případnou realizaci.

V závislosti na tom, jak se tým rozhodne, by mohla tato bakalářská práce v budoucnu sloužit oddělení EOA společnosti Škoda auto ke zlepšení stávajícího způsobu zajištění kvality zákaznického kódu.

Seznam použité literatury

Citace

[1] MEIJS, B., A. KROUWELS, W. HEUVELMANS, R. SOMMEN, 2004. *Enhancing the Quality of ABAP development.* 1st ed. Walldorf: Galileo Press, 2004. ISBN 1-59229-030-2.

[2] MASSEN, A., M. SCHOENEN, D. FRICK, A. GADATSCH, *SAP R/3 Kompletní průvodce.* Brno: Computer Press, a.s., 2007. ISBN 978-80-251-17750-7.

[3] SAP R3 Architecture Introduction. *SAP Online Tutorials.* [online]. © 2013 – 2014 [cit. 2015-04-21]. Dostupné z: *http://sapbrainsonline.com/help/sap-r3-architecture-introduction.html*

[4] Interní materiály společnosti ŠKODA AUTO a.s.

[5] LINKIES, M., H. KARIN. *SAP Security and Risk Management.* 2nd ed. Galileo Press, 2011. ISBN 978-1-59229-355-1.

[6] Introduction to SAP ABAP Program types. *SAP Community Network.* [online]. 1.6.2012 [cit. 2015-04-05]. Dostupné z: *http://scn.sap.com/docs/DOC-28783*

[7] Code Inspector. *Sap Community Network.* [online]. 21.3.2014 [cit. 2015-03-15]. Dostupné z: *http://wiki.scn.sap.com/wiki/display/ABAP/Code+Inspector*

[8] ABAP Test Cockpit – an Introduction to SAP's new ABAP Quality Assurance Tool. *SAP Comunnity Network.* [online]. 19.9.2012 [cit. 2015-03-20]. Dostupné z: *http://scn.sap.com/docs/DOC-31773*

[9] The Quality of your ABAP Development. *Information Technology Partners, INC.* [online]. 12.4.2014 [cit. 2015-03-21]. Dostupné z: *http://www.itpsap.com/blog/2014/04/12/the-quality-of-your-abap-development-part-1/*

[10] The Virtual Forge Code Profiler. *Virtual Forge.* [online]. © 2014 [cit. 2015-04-01]. Dostupné z: *https://www.virtualforge.com/en/portfolio/codeprofiler.html*

[11] THOMASIS, T. D., A. TEMPLETON. *Managing Custom code in SAP*. 1st ed. Boston: Galileo Press, 2013. ISBN 978-1-59229-436-7.

Bibliografie

KÜHNHAUSER, K. H. *ABAP Výukový kurz.* Brno: Computer press, a.s., 2012. ISBN 978-80-251-2117-7.

Kvalita softwaru. *FCC PUBLIC.* [online]. © 2015 [cit. 2015-03-13]. Dostupné z: *http://www.odbornecasopisy.cz/index.php?id_document=31468*