Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Testování softwaru automatických převodovek osobních automobilů

Jiří Konvář

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jiří Konvář

Informatika

Název práce

Testování softwaru automatických převodovek osobních automobilů

Název anglicky

Automatic transmission software testing for passenger cars

Cíle práce

Téma diplomové práce se zabývá problematikou testování softwaru v oblasti automobilového průmyslu, konkrétně automatických předovek osobních automobilů ve společnosti ZF Engineering Plzeň. Hlavním cílem práce je analýza kvality současných procesů testování a návrh optimalizace procesů testování ve společnosti ZF Engineering Plzeň.
Dílčí cíle jsou:
1. charakterizovat problematiku oblast testování softwaru,
2. analyzovat současné procesy v oblasti testování softwaru automatických převodovek osobních automobilů ve vybrané společnosti,
3. navrhnout řešení optimalizace kvality procesů v oblasti testování softwaru automatických převodovek osobních automobilů.

Metodika

Teoretická část diplomové práce se bude zabývat teoriemi a metodikami obecného testování softwaru a dále budou představeny procesy spojené s testováním v automobilovém průmyslu. Výstupem praktické části bude návrh optimalizace procesů testování softwaru automatických převodovek osobních automobilů na základě teoretických poznatků a provedené analýzy ve společnosti ZF Engineering Plzeň v současné době.

Doporučený rozsah práce

60–80 stran

Klíčová slova

ZF Engineering Plzeň, testování softwaru, softwarový tester, nástroj exam, manuální testování, automatické testování, HIL, životní cyklus software, převodovka automobilu, automobilový průmysl

---

Doporučené zdroje informací

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

DOLEŽAL, Jan, Pavel MÁCHAL a Branislav LACKO. Projektový management podle IPMA. 2., aktualiz. a dopl. vyd. Praha: Grada, 2012. Expert (Grada). ISBN 978-80-247-4275-5.

ISTQB, Certified Tester Foundation Level Syllabus Version 2018 V3.1

MYSLÍN, J. Procesy vývoje software. Praha: Vysoká škola podnikání a práva, a.s., 2016.

PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 14. 7. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 12. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Testování softwaru automatických převodovek osobních automobilů" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31. března 2023 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Michalovi Stočesovi, Ph.D. za odborné vedení mé diplomové práce a společnosti ZF Engineering Plzeň s.r.o. za poskytnutí potřebných informací.

Testování softwaru automatických převodovek osobních automobilů

Abstrakt

Diplomová práce se zaměřuje na problematiku testování softwaru v oblasti automobilového průmyslu, konkrétně automatických převodovek osobních automobilů ve společnosti ZF Engineering Plzeň. Hlavním cílem práce je analýza kvality současných procesů testování a návrh optimalizace procesů testování ve společnosti ZF Engineering Plzeň.

Teoretická část se zabývá představením teorie vývoje softwaru, do které patří metodiky vývoje softwaru a popis testování softwaru obecně. Dále teoretická část popisuje techniky a management testování.

V úvodu praktické části je představena konkrétní společnost, ve které je charakterizována problematika testování softwaru. Na základě analýzy současných procesů je proveden návrh optimalizace procesů testování softwaru. V závěru praktické části je provedena implementace vycházející z navrhovaných opatření pro řešení procesů, která je doplněna o výsledky z měření. Výsledky mohou sloužit jako podklady pro zavedení navrhovaných optimalizací do standartních procesů testování softwaru automatických převodovek osobních automobilů.

Klíčová slova: ZF Engineering Plzeň, testování softwaru, softwarový tester, nástroj exam, manuální testování, automatické testování, HiL, životní cyklus software, převodovka automobilu, automobilový průmysl

Automatic transmission software testing for passenger cars

Abstract

The thesis examines software testing in the automotive industry, focusing specifically on automatic transmissions of passenger cars in the company ZF Engineering Plzeň. The main aim of the thesis is to analyse the quality of current testing processes and to propose solutions for the optimization of these processes in ZF Engineering Plzeň.

The theoretical part introduces the theory of software development, which includes software development methodologies and a description of software testing in general. It also describes testing techniques and management.

The practical part starts with the introduction of a specific company describing the issue of software testing. Based on the analysis of current practices, a proposal for the optimization of software testing processes is made. The practical part concludes with an implementation based on the proposed measures, which is supported by the results from the measurements. The outcomes can serve as a basis for introducing the proposed optimizations into standard automotive automatic transmission software testing processes.

Keywords: ZF Engineering Plzeň, software testing, software tester, exam tool, manual testing, automatic testing, HiL, software life cycle, automotive transmission, automotive industry

Obsah

1 Úvod **12**

2 Cíl práce a metodika **13**

2.1 Cíl práce 13

2.2 Metodika 13

3 Teoretická východiska **14**

3.1 Úvod do testování softwaru 14

3.2 Metodiky vývoje softwaru 14

3.3 Tradiční metodiky 15

3.3.1 Vodopádový model životního cyklu 15

3.3.2 Spirálový model životního cyklu 17

3.3.3 V-model 18

3.4 Agilní metodiky 20

3.4.1 SCRUM 22

3.4.2 Extrémní programování 24

3.4.3 Feature Driven Development 26

3.4.4 Test Driven Development 27

3.5 Testování 28

3.5.1 Základní cíle testování 28

3.5.2 Principy testování 29

3.5.3 Proces testování 31

3.5.4 Úrovně testování 35

3.5.5 Psychologie testování 39

3.6 Techniky testování 40

3.6.1 Typy testů 40

3.6.2 Statické testování 42

3.6.3 Testování černé skříňky 47

3.6.4 Testování bílé skříňky 49

3.6.5 Testování založené na zkušenostech 50

3.7 Management testování 51

3.7.1 Plánování testování 52

3.7.2 Monitoring testování 55

3.7.3 Rizika testování 56

3.7.4 Management defektů 58

4 Vlastní práce **62**

4.1 Představení společnosti 62

4.1.1 Popis automatické převodovky 63

4.1.2 Současný proces testování v ZF Engineering Plzeň s.r.o. 66
4.1.3 Nástroje a hardware k testování 71
4.2 Aktuální stav manuálního testování 75
4.2.1 Procesy komponentního testování 75
4.2.2 Příprava specifikace testování 76
4.2.3 Realizace testování 77
4.2.4 Zpracování výsledku testování 78
4.3 Aktuální stav automatického regresního testování 79
4.3.1 Procesy automatického testování 80
4.3.2 Příprava specifikace testování 80
4.3.3 Realizace automatického testování 81
4.3.4 Zpracování výsledků automatického testování 82
4.4 Analýza současného procesu manuálního testování 82
4.4.1 Analýza procesů komponentního testování 83
4.4.2 Analýza přípravy specifikace testování 84
4.4.3 Analýza realizace manuálního komponentního testování 86
4.4.4 Analýza zpracování výsledku manuálního komponentního testování 88
4.5 Analýza současného procesu automatického regresního testování 89
4.5.1 Analýza procesů automatického testování 89
4.5.2 Analýza přípravy automatického regresního testování 90
4.5.3 Analýza realizace automatického regresního testování 90
4.5.4 Analýza zpracování výsledků automatického testování 91
4.6 Návrh opatření řešení procesu manuálního testování 91
4.6.1 Navrhované řešení procesů komponentního testování 92
4.6.2 Navrhované řešení přípravy specifikace testování 93
4.6.3 Navrhované řešení realizace manuálního komponentního testování 94
4.6.4 Navrhované řešení zpracování výsledků manuálního komponentního testování 95
4.7 Návrh opatření řešení procesu automatického regresního testování 96
4.7.1 Navrhované řešení procesů automatického regresního testování 96
4.7.2 Navrhované řešení přípravy automatického regresního testování 97
4.7.3 Navrhované řešení realizace automatického regresního testování 97
4.7.4 Navrhované řešení zpracování výsledků automatického regresního testování 97

5 Výsledky a diskuse 99
5.1 Dosažené výsledky po implementaci automatického regresního testování 99
5.1.1 Výsledky automatického regresního testování po kalibrační revizi 99
5.1.2 Výsledky automatického regresního testování po korekci softwarových požadavků 102
5.2 Porovnání výsledků implementace 105

6 Závěr **106**

7 Seznam použitých zdrojů **107**

8 Seznam obrázků, tabulek, grafů a zkratek **110**

8.1 Seznam obrázků 110

8.2 Seznam tabulek 110

8.3 Seznam grafů 110

8.4 Seznam použitých zkratek 111

Příloha A **112**

1 Úvod

V současné době proniká software do všech oblastí podnikání, dopravy, průmyslu, kde se jeho potenciál a růst neustále zvyšuje. Stále více se stává součástí našich životů, jelikož se nachází v celé řadě chytrých zařízení, se kterými jsme úzce v kontaktu v našich domácnostech. Softwaru nejprve předchází jeho samotný vznik, který je určený životním cyklem vývoje softwaru.

Na nově vznikající software je v současnosti kladen stále větší důraz v kvalitě a spolehlivosti. Pro vznik kvalitního a spolehlivého softwaru je klíčové důsledné testování v průběhu životního cyklu vývoje softwaru. Testování softwaru bývá často opomíjeno z důvodu mylných představ zdlouhavosti, zbytečnosti, nudnosti či z finančních nákladů. Jedná se o chybný úsudek, jelikož odhalení defektů až po nasazení softwaru do běžného provozu může způsobit velké finanční škody, poškodit pověst podniku nebo dokonce ohrozit zdraví či životy osob. Čím dříve v průběhu vývoje softwaru dojde k odhalení neočekávaného chování způsobené defekty, tím je jejich odstranění méně nákladnější a jednodušší. Je důležité si uvědomit, že testování nezaručuje bezchybnost softwaru; je nutné si připustit určité procento chyb. Pokud se v průběhu testování neodhalí defekty, neznamená to, že je software bez chyb. Rozsáhlost, funkce a účel použití softwaru určuje, jak důkladně bude testování softwaru provedeno.

Pro testování softwaru je důležité určit hranici, kdy se ještě vyplatí testovat a kdy již nikoliv, i na úkor nenalezených defektů. Důkladnější testování vyžaduje větší časovou náročnost a vyšší finanční náklady. Neměl by nastat stav, kdy testování bude časově náročnější a finančně nákladnější než samotný vývoj softwaru. Je tedy nutné průběžně analyzovat procesy vývoje a testování softwaru a pracovat na jejich optimalizaci, aby bylo dosaženo co nejefektivnějšího vývoje a testování softwaru, na jehož výstupu bude dodán kvalitní, spolehlivý a výkonný software, který bude splňovat požadavky zákazníka. Diplomová práce se podrobně zabývá optimalizací procesů testování softwaru.

2 Cíl práce a metodika

2.1 Cíl práce

Téma diplomové práce se zabývá problematikou testování softwaru v oblasti automobilového průmyslu, konkrétně automatických převodovek osobních automobilů ve společnosti ZF Engineering Plzeň. Hlavním cílem práce je analýza kvality současných procesů testování a návrh optimalizace procesů testování ve společnosti ZF Engineering Plzeň.

Dílčí cíle jsou:

1. charakterizovat problematiku oblasti testování softwaru,
2. analyzovat současné procesy v oblasti testování softwaru automatických převodovek osobních automobilů ve vybrané společnosti,
3. navrhnout řešení optimalizace kvality procesů v oblasti testování softwaru automatických převodovek osobních automobilů.

2.2 Metodika

Teoretická část diplomové práce se bude zabývat teoriemi a metodikami obecného testování softwaru a dále budou představeny procesy spojené s testováním v automobilovém průmyslu. Výstupem praktické části bude návrh optimalizace procesů testování softwaru automatických převodovek osobních automobilů na základě teoretických poznatků a provedené analýzy ve společnosti ZF Engineering Plzeň v současné době.

3 Teoretická východiska

3.1 Úvod do testování softwaru

Softwarové systémy se v současné době vyskytují všude kolem nás a pronikají do všech odvětví. Pomalu se stávají neoddělitelnou součástí našeho života. Software obsahují rozsáhlé podnikové informační systémy, bankovní systémy, dále zařízení, jako jsou telefony a televize, dále je možné se s nimi setkat ve strojích a dopravních prostředcích. Všechny tyto produkty a zařízení, i přestože jsou z různých a odlišných oborů mají jedno společné a tím je správně fungující software.

Nefunkční software má za následek mnoho problémů týkající se finanční a časové ztráty nebo dobré pověsti podniku, a v krajních případech může způsobit dokonce zranění či smrt. Testování softwaru je dnes důležitou součástí procesu vývoje softwarových systémů, který obsahuje celou řadu různých aktivit. Jedná se o důležitou činnost, která posuzuje kvalitu softwaru a snižuje pravděpodobnost selhání softwaru v provozu. Testování softwaru je závislé především na modelu životního cyklu vývoje softwaru. (1, s. 12). Hlavním cílem testování softwaru je vyhledávat chyby a defekty, a ne zjišťovat, zda software funguje. Testování by mělo chyby v softwaru odhalit v prvotních fází, jelikož náklady vynaložené s opravou chyb jsou výrazně nižší, než odhalení chyb v pokročilých fází vývoje softwaru nebo dokonce při předání hotového softwaru, které mohou překročit únosnou mez. Veškeré metodiky vývoje softwaru, rozdělení testování, procesy s ním spojené a cíle testování budou popsány v následujících kapitolách.

3.2 Metodiky vývoje softwaru

Pro testování softwaru je klíčová znalost metodiky na základě, které se software vyvíjí. Důležité je si uvědomit, že testování softwaru není izolovanou formu, jde o součást životního cyklu vývoje softwaru (2, s. 24). Model životního cyklu vývoje softwaru je proces, podle kterého se vytváří systémový produkt, a to od prvotní iniciace až po konec životnosti softwaru (3, s. 27). „Metodika je souhrn vodítek a principů, které mohou být přizpůsobeny a aplikovány na řešení určité situace. Může jít jak o jednoduchý soupis akcí, které se mají vykonat, tak o specifický přístup, šablony, formuláře, popřípadě kontrolní seznamy“ (4). Metodiky vývoje softwaru se rozdělují do dvou základních skupin na klasický přístup a agilní přístup.

3.3 Tradiční metodiky

Tradiční metodiky jsou řízeny přesnými postupy dle standardizace procesů, ve kterých je cílem, co nejpečlivěji analyzovat přesné požadavky zákazníka a určit jednotlivé termíny vyhotovení. Dále jsou zde přesně dané role jednotlivých členů týmu, kteří zodpovídají za plnění svých povinností. Tradiční metodiky nachází své uplatnění především v projektech větších rozsahů, na kterých spolupracuje více velkých týmů, které mají geologicky rozdělené pobočky, které mají velké korporátní společnosti. Dále je vhodné také pro projekty, u kterých je nutné dodržet přesně funkcionalitu, termíny dokončení či finanční rozpočet. Klíčovými vlastnostmi tradičních metodik je řád, pořádek, harmonogram a jistota. Nevýhodou tradičních metodik je zdlouhavý proces vývoje softwaru, který není možné během průběhu vývoje měnit. Hlavním důsledkem prodloužení procesu vývoje softwaru je důraz na analýzu požadavků zákazníka, komunikaci se zákazníkem, precizní a rozsáhlou dokumentaci a administrativu. Další nevýhodou je, že zákazník dlouhou dobu nedostává žádné dílčí výsledky v průběhu vývoje softwaru a nemůže se tedy vyjadřovat ke změnám během vývoje (5, s. 64-66).

Do tradičních metodik patří:

- Vodopádový model
- Spirálový model
- V-model, W-model
- RUP
- PRINCE2

3.3.1 Vodopádový model životního cyklu

Vodopádový model patří k nejstarším, významným a základním modelům v oblasti řízení vývoje softwaru, který vznikl na počátku sedmdesátých let. Jedná se o model, který vyniká svojí jednoduchostí, elegancí, přehledností a při aplikaci ve vhodných projektech je dobře funkční. Vodopádový model se hojně využívá při výuce programování jako velice názorný a přehledný model řízení projektu vývoje softwaru. Pojmenování Vodopádový model si vysloužil tento název, který vystihuje jeho chování při řízení projektu, jelikož jednotlivé fáze projektu jsou prováděny postupně od myšlenky na produkt až po finální produkt v daném pořadí s minimálními iteracemi, stejně jako tekoucí voda hora dolů ve vodopádu, jak je znázorněno v příloze A na obrázku 15 Schéma Vodopádového modelu

životního cyklu softwaru. Vývoj projektu je řízený postupně. Po končení jedné fáze může být zahájena následující fáze a není možné, aby se provádělo více fází současně. Na konci každé fáze je provedeno zhodnocení, zda je možné pokračovat do další fáze, pokud nesetrvá ve stejné úrovni, dokud nejsou požadavky splněny a fáze není dokončena. Ve vodopádovém modelu je kladen důraz na přesné požadavky a specifikaci výsledného produktu, jelikož není možné se vracet do předchozích fází. Pokud tedy nebude správně definován výsledný produkt, je riziko, že zákazníkovi bude dodán nevhodný produkt (6).

Vodopádový model se může rozdělit do jednotlivých rámců, které reprezentují jednotlivé činnosti, jak je patrné na obrázku 1 Rámec vodopádového řízení projektu.

INICIACE PROJEKTU je významným bodem v začátcích vývoje softwaru, jelikož se zjišťuje, zda má projekt vůbec smysl realizovat či nikoliv. Představuje pomyslné síto, které odděluje zrno od plev.

PLÁNOVÁNÍ PROJEKTU je proces, při kterém dochází k sestavování plánu a modelování průběhu realizace projektu. Plánování je nutnost, bez které se projekt nemůže obejít. Během plánování je důležité se vyvarovat vysoce preciznímu plánování, jelikož plán modelu je vždy částečně zkreslený oproti skutečnosti a projekt se tím následně stává nereálným. Plánování se upřesňuje dle jednotlivých činností na plány kvalit, plány času a plány nákladů.

SLEDOVÁNÍ PROJEKTU je zajištěno pomocí důležitého nástroje, kterým je Směrný plán. Pomocí tohoto nástroje se vyhodnocuje průběh projektu. Dochází zde k evidenci vykonané práce a zároveň srovnáním skutečnosti a plánu, které vypovídá o průběhu vývoje projektu. V případě velkých rozdílů je nutné projekt aktualizovat.

ŘÍZENÍ PROJEKTU je klíčové k řízení a celkovému uřízení projektu, jelikož projekty neprobíhají vždy podle plánu, protože bývají často nepřesné či nadsazené. Je tedy nutné co nejpřesněji kopírovat průběh projektu dle plánu.

UKONČENÍ PROJEKTU je dosažením samostatného výstupu finálního produktu projektu. V této části dochází k podpisu finálního akceptačního protokolu a provádí se uzávěrkové operace. Na řadu přichází zhodnocení projektu a identifikace důležitých zkušeností, které je možné využít do nadcházejících projektů. Neméně důležité je hodnocení výkonu týmu, který projekt realizoval. Zpravidla existují dva druhy ukončení projektů. První je standartní ukončení projektu, jehož výsledkem je finální produkt a druhým je předčasné ukončení projektu, které bylo zapříčiněno negativními vlivy v průběhu projektu a výsledkem je nedokončený produkt (7, s. 23-56).

Výhodou tohoto modelu je jeho přehlednost v jednotlivých fázích, přímočarý vývoj, každý člen přesně ví, co má dělat a jednoduchost kontroly zejména z hlediska řízení firmy. Nevýhoda modelu je velká jednoduchost k řízení rozsáhlých a složitých projektů. Vodopádový model je nevhodné použít v projektech, které vyžadují pružnost při zavádění změn v průběhu vývoje projektu. Další nevýhoda tohoto modelu je v oblasti testování, i přestože je vše pečlivě specifikováno, důkladně popsáno a testovací tým je seznámen, jak má software testovat a jaký by měl výsledek. Probíhá testování až v závěrečné fázi vývoje softwaru, když je software již zcela dokončený. Není zde docíleno, že chyby v softwaru je nejvhodnější analyzovat již v raném vývoji softwaru a je zde riziko, že náklady projektu neúměrně stoupnou. Vodopádový model je vhodný pro menší týmy, které se zaměřují na řízení rozsahem menších specifikací jednoduchých projektů (3, s. 30).

Obrázek 1 Rámec vodopádového řízení projektu

Zdroj: Dvořák (2017) s. 23, obr. 8.

3.3.2 Spirálový model životního cyklu

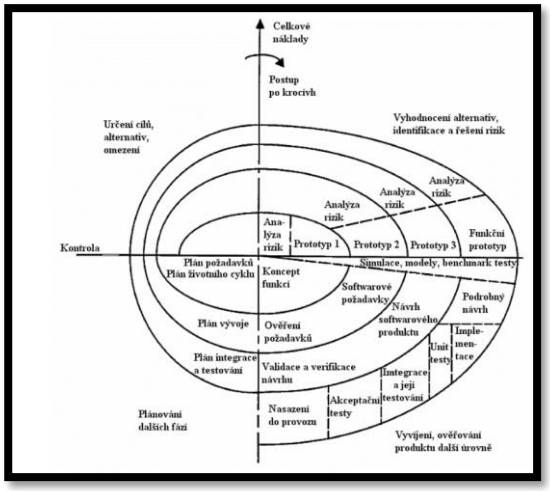
Spirálový model životního cyklu vývoje softwaru vznikl v roce 1986 a jeho autorem je Barry Boehm. Tento model vznikl na základě odstranění základního problému vodopádového modelu a tou je jeho čistá linearita, která je ve složitějších projektech nežádoucí a představuje úskalí při optimalizaci v průběhu projektu.

Spirálový model oproti vodopádovému modelu, kde je nutné znát detailně všechny požadavky produktu, se liší zásadní myšlenkou, ve které není nutné přesně a dopodrobna definovat výsledný produkt, ale definovat pouze základní a důležitou část a dále přejít do další fáze. Celý proces se opakuje tak dlouho, dokud není docíleno výsledného produktu, jak je znázorněné na obrázku 2 Schéma spirálového modelu životního cyklu softwaru. Spirálový model obsahuje spirálu, která prochází čtyřmi základními

kvadranty. Je možné se také setkat s podrobnějším spirálový model, který je rozšířen na 6 cyklů. Výše nákladů je určena pomyslnou osou Y a jejich růst je znázorněn kruhy spirály. Čím víc je cyklů opakování, tím rostou náklady na výsledný produkt.

Hlavní výhodou spirálového modelu je možnost změn požadavků zákazníka v průběhu vývoje softwaru, průběžná analýza rizik a klíčové je zejména pravidelné testování již v počátcích a v průběhu projektu, což je důležité pro pravidelné odhalení chyb a nižší náklady při jejich odstranění. Nevýhodou je složitost oproti vodopádovému modelu a důkladná znalost této metodiky. Dále je pak kladen velký důraz na kvalitu vývojářského týmu, který musí správně provést analýzu rizik. Rychlost průběhu vývoje je zpomalena důkladnou administrativní činností. Spirálový model je vhodný k použití pro rozsáhlé a komplexní projekty, ve kterých dochází k častým změnám v průběhu projektu a je nutná intenzivnější součinnost se zákazníkem v průběhu vývoje. V současné době je tato metodika používána méně, protože již existuje celá řada propracovaných metodik (8).

Obrázek 2 Schéma spirálového modelu životního cyklu softwaru



Zdroj: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model

3.3.3 V-model

V-model životního cyklu vývoje softwaru vznikl v druhé polovině osmdesátých let. Tento model vychází z principu vodopádového modelu, který byl rozšířen o druhou větev a bylo tak docíleno tvaru písmena V, jak je patrné z jeho názvu. V příloze A je na obrázku 16 Schéma V-modelu životního cyklu softwaru je vyobrazena struktura modelu. Podnětem

vzniku V-modelu byla aktivita ze strany testerů pro zvýšení významu testování ve srovnání s návrhem a programováním softwaru.

Principem V-modelu je struktura, která je složena z několika částí. Levá strana obsahuje aktivity, které souvisejí se specifikací a návrhem. Pravá strana se zaměřuje na aktivity dynamického testování. Obě strany jsou mezi sebou propojeny a tvoří spolu dvojici aktivit, které jsou:

- specifikace požadavků – uživatelské akceptační testy
- hrubý návrh – systémové testy
- detailní návrh – integrační testy

Důležitost testování je docílena vazbou k jednotlivým požadavkům a návrhům, ke kterým jsou přiřazené určité typy testů. Špička ve spodní části V-modelu zastupuje programování, při kterém zároveň dochází k jednotkovému testování. Dále může být V-model pomyslně horizontálně rozdělen na dvě poloviny dle odpovědnosti. Horní polovina zastupuje odpovědnost odběratele a spodní polovina je zaměřena na odpovědnost dodavatele nebo oddělení vývoje.

Jak je z V-modelu patrné není dostatečné pouze projekt správně navrhnout a vytvořit, ale je důležité jednotlivé úrovně podrobit testování, které se uvádí jako úrovně testování. Dle osobních zkušeností odborníků v oblasti řízeného testování vznikají nejúspěšnější projekty s pomocí oddělené úrovně testování, kde jsou s tím spojené odlišné cíle a odpovědnosti. Nevyplývá z toho, že by byla snížena důležitost testování v procesu programování. Identifikace chyb přímo programátory je velice důležitá, jelikož se jedná o levnější a rychlejší řešení než později prostřednictvím testerů.

Skutečnost často bohužel bývá odlišná oproti V-modelu. Namísto testování v určitých úrovní dochází k testování většinou až v závěru projektu, zapříčiněné velkým zpožděním v průběhu projektu. Není to způsobeno chybou V-modelu, ale spíše řízením manažerů nebo přístupem vývojářů. Nevýhody V-modelu jsou zejména v přímočarosti a jednoduchosti ve vztahu mezi jednotlivými úrovněmi návrhu a testování. Ve skutečnosti je někdy nedostačující testování pouze s jedním druhem požadavků a také není dosaženo přímé vazby s požadavky zákazníka. Další nevýhodou je, jak již bylo zmíněno výše v textu, pozdní přístup testerů k testování, jelikož to V-model umožňuje (9). Uvedené nedostatky V-modelu byly eliminovány vylepšeným již méně známým W-modelem. W-model klade důraz na včasné zapojení testerů již v úvodu vývoje softwaru, kdy se specifikují požadavky a vznikají jednotlivé návrhy. Tuto část znázorňuje levá strana modelu, ve které dochází

k otestování specifikace, hrubého návrhu a dále pak testování detailního návrhu a zároveň probíhá testování na pravé straně W-modelu. Rozšířená verze W-modelu již umožňuje zapojení testerů oproti klasickému V-modelu již v rané fázi od statického testování jednotlivých fází, až na úroveň integračních, systémových a dynamických testů. Dále jsou s modelem vázány další typy testů, jako jsou bezpečnostní, průzkumné, výkonnostní testy či testy použitelnosti a další. Uvedené typy testů nejsou vázány přesně s danou úrovní, ale je možné je vykonávat libovolně (10). V příloze A na obrázku 17 Schéma W-modelu životního cyklu softwaru je vyobrazena struktura W-modelu.

3.4 Agilní metodiky

Agilní metodiky se odlišují interaktivním přístupem vývoje softwaru na rozdíl od tradičního přístupu vývoje softwaru. Hlavní myšlenkou je oprostit se od složitých a zdlouhavých postupů, striktních procesů a vyčerpávající dokumentace v zájmu rychlosti vývoje softwaru. Oproti tradičním metodikám se agilní metodiky hodí spíše pro projekty menších rozsahů, na kterých pracuje jeden menší tým ve stejné geografické pobočce, jedná se tedy spíše o menší společnosti, které vyvíjí software. V agilních metodikách není smyslem zdlouhavá analýza požadavků zákazníka, dokonalá dokumentace a vyčerpávající administrativa, které urychlují celkový vývoj softwaru. Důležité je, aby vývojový tým nesklouzl k tomu, že nebude vytvářet dokumentaci, to není smyslem agilního přístupu vývoje softwaru. Agilní přístup je vhodný pro projekty, kde zákazník nemá přesně rozmyšlené požadavky, jak bude výsledný produkt vypadat a jeho podoba se vytváří v průběhu vývoje softwaru nebo pro projekty, které vyžadují časté změny ze strany zákazníka. Hlavním cílem je zákazníkovi předávat pravidelně malé části vytvořeného produktu, aby se k nim mohl vyjádřit a popřípadě navrhnout změny či úpravy. Základem agilního přístupu je Agilní manifest (Agile Manifesto), který vznikl v roce 2001. Nejedná se o žádný oficiální dokument, ale je to spíše prohlášení, které vzniklo na základě nespokojenosti s postupy v tradičních metodikách. Agilní manifest byl sepsán skupinou vývojářů, kteří byli přesvědčeni, že software lze vyvíjet odlišným způsobem než doposud (11).

Autoři agilního manifestu: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland a Dave Thomas (12, s. 39).

Formulace agilního manifestu:

„Objevujeme lepší způsoby vývoje software tím, že jej tvoříme a pomáháme při jeho tvorbě ostatním.

Při této práci jsme dospěli k těmto hodnotám:

- Jednotlivci a interakce před procesy a nástroji
- Fungující software před vyčerpávající dokumentací
- Spolupráce se zákazníkem před vyjednáváním o smlouvě
- Reagování na změny před dodržováním plánu
- Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.

Agilní manifest zahrnuje čtyři základní principy a principy agilního vývoje softwaru:

1. Jednotlivci a interakce před procesy a nástroji
2. Fungující software před vyčerpávající dokumentací
3. Spolupráce se zákazníkem před vyjednáváním o smlouvě
4. Reagování na změny před dodržováním plánu

Kromě základních čtyř principů obsahuje dalších dvanáct principů, které více rozšiřují a vysvětlují principy agilního manifestu.

Dvanáct principů:

1. Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
4. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
5. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
6. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
7. Hlavním měřítkem pokroku je fungující software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.

10. Jednoduchost--umění maximalizovat množství nevykonané práce je klíčová.
11. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
12. Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším a následně koriguje a přizpůsobuje své chování a zvyklosti“ (13).

Známé metodiky, které zastupují agilní přístupy vývoje softwaru:

- SCRUM
- Extrémní programování
- Feature Driven Development
- Test Driven Development

3.4.1 SCRUM

SCRUM patří ke známým a dosti využívaným agilním metodikám. Se SCRUM se pojí celá řada pojmů, artefaktů, skupin a rolí, které definují celou metodiku. Jedná se o agilně – objektově orientovanou metodiku, která se vyznačuje flexibilitou, iterativním a inkrementálním přístupem procesu vývoje softwaru. Metodika zahrnuje také úzký kontakt mezi členy vývojového týmu a zákazníkem. Lidé, kteří se v metodice SCRUM podílejí na vývoji softwaru, se dělí na dvě základní skupiny. První skupinou jsou Pigs a druhá skupina Chickens. Do první skupiny Pigs patří lidé, kteří jsou přímo členové týmu a jsou zahrnuti v projektu a přímo se podílejí na projektových prací a jsou součástí projektového týmu, a hlavně zodpovídají za výsledek projektu. Ve druhé skupině Chickens se nachází ostatní lidé, kterých se projekt týká, ale nejsou do něj přímo zapojeni, nemohou přímo projekt ovlivnit a nejsou zodpovědní za výsledek projektu (14).

Skupina Pigs je projektový tým, který definuje tři role:

- Product Owner: jedná se o důležitého člena týmu, který zastupuje zákazníka. Může se jednat o osobu interní nebo externí, která je přímo zaměstnancem zákazníka, který produkt poptává. Určuje vize, definuje požadavky zákazníka, funkcionalitu softwaru a má představu o tom, jak bude výsledný produkt vypadat. Dále neustále komunikuje se zákazníkem o vývoji projektu a konzultuje případné změny. V neposlední řadě také nese plnou odpovědnost za úspěch projekt.
- SCRUM Master: je druhá role projektového týmu, která vykonává manažerskou roli zodpovídající za technickou stránku týmu. Není to klasická manažerská role, jedná se spíše o podporu celého týmu, který se v podstatě řídí sám. SCRUM Master pomáhá týmu po metodické stránce dosáhnout cílů, má na starosti koordinaci

meetingů a přípravu dokumentace. Řeší problémy, spory, které vznikají během vývoje, má na starosti celkovou komunikaci mezi členy a zákazníkem. Dalším úkolem je motivace, podpora členů týmu a ochrana před rušivými vlivy, které odvádějí pozornost od definovaného cíle.

- SCRUM Team Member: je třetí a zároveň poslední rolí projektového týmu, která se skládá z klasických členů. Nejsou zde přesně rozdělené role, kdo co má, dělat. Všichni členové jsou všestranní, vyžaduje se od nich schopnost řízení samořízení a výběr úkolů si stanovují dle obtížnosti a zkušenosti (15).

Skupina Chickens obsahuje další tři role:

- Koncový uživatel
- Člen vedení organizace zákazníka
- Konzultanti

Jak již bylo uvedeno, metodika SCRUM obsahuje celou řadu pojmů a artefaktů:

- User Story: v metodice se jedná o velmi důležitý pojem, protože formuluje požadavky a definuje cíle zákazníka. Jedná se o uživatelský popis, jak by se měl software chovat a co by měl dělat, ale není zde uvedeno, jak by se vývoj softwaru měl realizovat.
- Sprint: jedná se o vývojovou iteraci, která se v metodice SCRUM uvádí pod názvem sprint. Délka sprintu není pevně stanovena; běžně trvá 30 dnů, ale délku je možné dle potřeby upravit. Na konci každého sprintu je zákazníkovi předvedena spustitelná a testovatelná část aplikace, která byla vytvořena v průběhu sprintu.
- Backlog: jde o další z důležitých pojmů v rámci metodiky, který uvádí dosud neimplementované user stories. Backlog lze jednoduše definovat jako seznam nesplněných a nevyřízených úkolů uživatelských popisů.

  Následně se backlog dělí na dvě části:

  1. Product Backlog: kompletní seznam úkolů user stories, který je nutné implementovat v rámci celého projektu.
  2. Sprint Backlog: je seznam úkolů, který se musí implementovat v rámci aktuálního sprintu.
- Meeting: v průběhu kompletní fáze vývoje probíhá celá řada schůzek, které jsou definovány dle tématu řešení.
- Meetingy plánovací: vzniká logický plán, ze kterého vychází Product Backlog a Sprint Backlog

- Meetingy hodnotící: hodnotí se zde celý projekt anebo jednotlivé sprinty
- Meetingy hodnotící i plánovací: Daily meeting, každodenní hodnocení z předchozího dne a plánování do dalšího dne (16, s. 3-5).

Ve SCRUM je vývoj rozdělen do tří fází.

1. Zahájení: počáteční fáze projektu, ve které se definují požadavky zákazníka, plánuje se spolupráce a rozdělují se role a kompetence ve vývojovém týmu.
2. Samotný vývoj: v této fázi nastává již samotný vývoj softwaru, který je iterativní a probíhá ve sprintech.
3. Ukončení: v poslední fází dochází k závěrečnému testování finálního softwaru a vývoj produktu je ukončen (17, s. 85-86).

Výhodou této metodiky je, že dokáže pružně reagovat na změny, které vznikají v průběhu vývoje softwaru. To je umožněno tím, že na konci každého sprintu dochází k otestování a akceptování nově vytvořené funkcionality ze strany vývojového týmu a ze strany zákazníka. Tím je zajištěno, že změny v projektu lze zapracovat po celou dobu vývoje softwaru a splnit tak požadavky zákazníka. Testování se provádí i na konci vývoje celého projektu. Nevýhodou metodiky SCRUM je, že zde nenachází klasický manažer a nejsou zde striktně rozdělené role dle činností jako v tradičních metodikách. Očekává se od vedoucí pozice určitá intuice, jelikož metodika spočívá spíše v doporučení procesů, není zde nic přesně dáno, jako např. délka sprintu. Dále se od členů projektového týmu vyžaduje určitá samostatná činnost, která spočívá ve výběrů úkolů dle svých schopností, pružnosti, flexibility a odpovědnosti. Metodika je vhodná spíše pro menší a vyspělé týmy, které realizují internetové projekty. Princip metodiky je znázorněn v příloze A na obrázku 18 Schéma životního cyklu metodiky SCRUM (16, s. 6-7).

3.4.2 Extrémní programování

Extrémní programování patří k rozšířené, známé a osvědčené agilní metodice, kterou lze použít v celé řadě projektů. Jak již z názvu vypovídá, používá ve svých procesech principy z jiných metodik, které využívá extrémním způsobem.
Extrémní programování vychází ze čtyř základních proměnných:

- Kvalita
- Čas

- Náklady
- Šíře zadání

První extrémní způsob je v délce iterace, která je u tradičních metodik dlouhá i několik měsíců, oproti tomu u extrémního programování je délka iterace počítaná ve dnech nebo dokonce jeden den a v extrémních případech i několikrát za den. Záleží na konkrétním případu vývoje, pokud je funkcionalita implementovaná a otestovaná, tak je možné ji nasadit. Dalším z extrémních způsobů je testování, které se provádí u tradičních metodik vždy na konci každé iterace, ale u extrémního programování se testují prakticky neustále dokola. Po návrhu nové funkce proběhne implementace, která se okamžitě otestuje a nečeká se několik dní na otestování.

V poslední řadě extrémů jsou změny, které jsou v podstatě v tradičních metodikách skoro nemožné. Oproti tomu extrémní programování je založené na změnách, které jsou prakticky neustále. Je možné, že i několikrát za den dojde ke změně specifikace a tím i ke změně kódu.

Z výše zmíněných změn vyplývají další čtyři hodnoty, které jsou pro extrémní programování klíčové:

1. Komunikace je nesmírně důležitá, protože musí komunikovat naprosto všichni mezi sebou a neexistuje zde oficiální meeting, ale komunikace je živelná a probíhá neustále.
2. Jednoduchost znamená, že se vyvíjí a dělá se jenom to, co je opravdu nutné, protože změny přicházejí velmi rychle a čím méně se vytvořená práce zahodí, tím to není tak velká ztráta. Je zde i absence zbytečné dokumentace a nepotřebných formulářů.
3. Zpětná vazba vzniká na základě testování, které probíhá neustále. Pokud testování odhalí chybu, dojde k její okamžité opravě a znovu se otestuje. Na testování se podílí i zákazník.
4. Odvaha spočívá v okamžitém řešení problémů, které nastaly, není zde prostor čekat pro absenci meetingu. Dále je nutná odvaha, a to konkrétně zahodit i několika týdenní práci a začít znovu než se snažit něco opravit, co již nemá smysl.

Na základě výše uvedených základních hodnot, extrémního programování využívá dalších dvanáct rozšiřujících postupů (18).

1. Zákazník jako součástí vývojového týmu
2. Párové programování

3. Stabilní čtyřicetihodinový pracovní týden
4. Týmové standarty
5. Společné vlastnictví a společná zodpovědnost
6. Plánovací hra
7. Malá verze
8. Metafora
9. Jednoduchý návrh
10. Refaktorizace
11. Nepřetržitá integrace
12. Testování je základ extrémního programování, kdy testy se navrhují ještě před samotným programováním tím, že se nejprve rozhodne, jak se ověří výsledek a na základě toho se vytvoří výsledek, který se testem otestuje.

Jednou z hlavních výhod extrémního programování je zejména velký důraz na testování, které prakticky probíhá neustále, z čehož vyplývá, že k odhalení chyb dochází již jak v začátcích projektu, tak i v přidávané funkčnosti. Další z velkých výhod je práce, která vyúsťuje v týmovou součinnost na základě lidských instinktů. Tím vzniká iterativní způsob vývoje softwaru s vysokým progresem. Tato výhoda přináší i úskalí, které přechází až do nevýhody této metodiky, při níž vzniká komplikované aplikování realizace v praxi pro všechny členy týmu, bez výjimky na roli v týmu. Další nevýhodou je nutná stála přítomnost zákazníka na pracovišti vývojového týmu. Metodika extrémního programování je vhodná pro menší týmy do deseti členů, které realizují menší projekty, jako jsou internetové systémy, ve kterých dochází k častým změnám požadavků od zákazníka (12, s. 68).

3.4.3 Feature Driven Development

Výše uvedená agilní metodika, zkráceně nazývaná FDD, je založena na základě vytvoření modelu, který charakterizuje celý systém produktu. Vytvořený model se převede do seznamu vlastností. Každému programátoru je v této metodice přidělena zodpovědnost za část doménového modelu neboli objekt, je to rozdíl oproti jiným agilním metodikám, kde si úkoly vybírají z nějakého seznamu, backlogu, anebo všichni zodpovídají za vše. Tímto způsobem vývoje se odlišuje od ostatních agilních metodik, která s tvorbou modelu nebo jiných inicializačních návrhů nepracují. Vývoj v metodice FDD je složena z pěti klíčových fází, kdy první tři fáze jsou sekvenční a zbylé dvě fáze jsou iterativní.

- Fáze 1 se zabývá tvorbou modelu, který vychází z diagramu tříd, většinou vytvořený pomocí jazyka UML.
- Fáze 2 spočívá ve vytvoření co možná nejpřesnějšího a podrobného seznamu jednotlivých vlastností získaných na základě vytvořeného modelu z předchozí fáze.
- Fáze 3 je poslední sekvenční fází a obsahuje vytvoření plánů, dle kterých dochází k postupné implementaci jednotlivých vlastností. V seznamech je uvedené pořadí implementace na základě priorit a vzájemných závislostí.
- Fáze 4 pracuje s návrhem, který obsahuje konkrétní vlastnosti, jejichž vybraná množina slouží k vytvoření a spuštění procesu DBF „Design By Feature“ návrh podle vlastností, který spočívá v určení tříd, které pokrývají dané vlastnosti a také zde dochází ke kontaktování jednotlivých vlastníků objektu. DBF zahajuje hlavní programátor. Na základě procesu DBF dochází k vypracování podrobného návrhu, dle kterého bude probíhat následná implementace.
- Fáze 5 spočívá již v implementaci, která je realizována prostřednictvím procesu BBF „Build By Feature“ realizace podle vlastností. Za návrh a implementaci každé třídy odpovídá její vlastník, který zároveň provádí její testování. Ve chvíli, kdy hlavní programátor akceptuje dosažený výsledek, dochází k integrování vlastnosti do aplikace. Souběžně pracuje více týmů, které zapracovávají a implementují nové vlastnosti. Složení týmů se může v průběhu procesu měnit. FDD je metodika, která neumožňuje tolik volnosti jako ostatní agilní metodiky, ale zavádí přidělení aktivity a zodpovědnosti na dané členy týmů. Je vhodná do týmů, které nedokážou pracovat v režimu například extrémního programování, ale potřebují do svého procesu vnést řád. Testování v metodice FDD přichází na řadu, vždy po implementaci každé vlastnosti, tím je docíleno včasné odhalení chyb. Spojením sekvenčních a iterativních fází vzniká dynamická metodika, která umožňuje reagovat na změny zákazníka a dodat software dle jeho přání (19).

3.4.4 Test Driven Development

Test Driven Development je agilní metodika, založená na testování softwaru, který je předmětem vývoje. Jedná se o neobvyklou metodiku, ve které je nutné nejprve napsat testy ještě před napsáním kódu. Teprve, když je napsaný testovací kód je možné začít psát zdrojový kód programu, který zavádí funkčnost systému. Při aplikaci této metodiky je možné implementovat takové množství kódu, které umožňuje otestování.

Hlavním principem Test Driven Development je tzv. refaktorizace, jedná se o úpravy zdrojového kódu bez změny funkčnosti, který se stále zjednodušuje a zefektivňuje, zároveň dochází i ke změně kódu testu. V prvotní fázi je nutné vytvořit takový test, který selže. Dále je možné provést po napsání selhávajícího testu implementační testovací funkci. Pokud je splněná prvotní fáze je možné pustit k otestování všechny vytvořené a zbylé testy. Jedná-li se o rozsáhlý projekt, je možné spustit pouze určitou podmnožinu testů. Nyní je možné přejít přímo k samotné tvorbě a popřípadě úpravě zdrojového kódu, následně se pak spustí nad nově vytvořeným programovým kódem všechny testy. Zdrojový kód se neustále upravuje, testuje a popřípadě se implementují nové funkce, dokud testování neproběhne úspěšně. Po kompletním úspěšném otestování je software připraven k dokončení. Agilní metodika Test Driven Development neklade důraz na proces a nezaměřuje se tolik na tvorbu specifikací, dokumentace a ostatní administrativní úkony. Metodika dále umožňuje tvůrčí volnost, která může způsobit, že některý dopsaný zdrojový kód, nebude pokrytý testem. V souvislosti s nižším důrazem na celkovou administrativu a tvůrčí volností je metodika vhodná pro týmy, které jsou složeny ze členů, kteří mají větší zkušenost, zodpovědnost, disciplínu a intuici, jelikož je nutné psát testy ke kódu, který dosud neexistuje a určit množství informací potřebných k vývoji, jako jsou např. požadavky zákazníka. Důležité je správné pochopení smyslu Test Driven Development. Testování není jen pro ověření správného chování softwaru, ale jedná se o princip celé metodiky. Velkou předností při správném pochopení metodiky je velice kvalitní výsledný software, jehož požadovaná funkčnost a chování je prověřeno důkladným testováním (20).

3.5 Testování

Tato kapitola bude již zaměřena zcela na testování, kde bude vysvětlen proces testování, popsány jednotlivé principy testování, představeny používané typy testů a na závěr bude zmíněna důležitost psychologie v oblasti testování.

3.5.1 Základní cíle testování

Mezi hlavní cíle testování softwaru pro předcházení vzniku defektů a chyb. Aby bylo možné chybám a defektům předcházet, je nutné analyzovat procesy vývoje softwaru, které jsou největšími zdroji chyb v průběhu vývoje softwaru. Příčiny vzniku chyb a defektů jsou různé. K největším a zásadním důvodům patří chyby ve specifikaci, kterých je celá řada. Často dochází k tomu, že specifikace není vůbec vytvořená. Dochází k tomu, že specifikace

je sice napsaná, ale není dostatečně podrobná, nebo se velice často mění. Dále nemusí být primárně chyba ve specifikaci, ale dochází k situacím, kdy specifikace je podrobná až vyčerpávající, ale členové vývojového týmu ji špatně porozumí nebo se s ní nedostatečně seznámí. K dalším neméně důležitým zdrojům defektů patří návrh. Návrh spočívá v popisu plánu softwaru, který realizují programátoři. Zdroje chyb v návrhu vznikají podobně jako u specifikace. Návrh bývá často uspěchaný nebo neustále dochází k jeho změnám a v neposlední řadě je návrh nedostatečně konzultován s ostatními členy týmu. Třetím velkým zdrojem defektů jsou chyby v programovém kódu. Příčiny chyb v kódování jsou podobné jako u specifikace a návrhu, kdy je nedostatečná dokumentace. Dále velmi časté změny kódu, které nesou riziko zanesení chyby a následně časová tíseň a programování pod tlakem. Zásadním zdrojem defektu v kódování je špatné porozumění programátora, jak danou funkcionalitu naprogramovat, která je způsobená právě chybou ve specifikaci, jak již bylo zmíněno v textu výše. Posledním zdrojem defektů jsou různé příčiny, které není možné specifikovat výše uvedenými zdroji. Může se jednat o kombinaci různých příčin a opakování stejných chyb. Dále jsou v nich zahrnuty i chyby v testování. Všechny ostatní příčiny tvoří malé procento chyb v celkovém objemu, ale je potřené je také do předcházení chyb a defektů zahrnout. Z této kapitoly vyplývá, že je nutné klást silný důraz na správnost specifikace, jelikož se může stát i zásadním zdrojem chyb v oblasti programového kódu (21).

3.5.2 Principy testování

Na základě praktických zkušeností a vývoje v oblasti testování softwaru v průběhu několika desítek let, byla definována celá řada principů, ze kterých nyní vychází obecný návod v oblasti testování softwaru. V současné době se uvádí sedm základních principů (22). „Sedm principů testování:

1. Testování ukazuje přítomnost defektů, nikoli jejich nepřítomnost.

   Testování může ukázat, že defekty jsou přítomny, ale nemůže prokázat, že žádné defekty neexistují. Testování snižuje pravděpodobnost, že v softwaru zůstaly neodhalené defekty, nicméně pokud nejsou žádné defekty nalezeny, není tím prokázána jeho správnost.
2. Kompletní testování není možné.

   Testování všeho (všech kombinací vstupů a vstupních podmínek) není možné s výjimkou triviálních případů. Místo snahy o kompletní testování je lepší se zaměřit na analýzu rizik, vhodné techniky testování a prioritizaci.

3. Včasné testování šetří čas a peníze.

   Pro včasné zjištění defektů by měly být statické i dynamické testovací aktivity zahájeny co nejdříve v životním cyklu vývoje softwaru. Včasné testování je někdy označováno jako shift left. Testování v rané fázi vývoje softwaru pomáhá snížit nebo eliminovat budoucí nákladné změny.

4. Shlukování defektů.

   Většinu defektů zjištěných během testování před vydáním obsahuje obvykle malé množství modulů nebo jsou defekty v těchto modulech zodpovědné za většinu provozních poruch. Předpokládané a skutečně pozorované shluky defektů během testování nebo v provozu jsou významným přínosem pro analýzu rizik, která se využívá k vhodnému zaměření testovacích aktivit (jak zmiňuje Princip 2).

5. Vyvarování se pesticidnímu paradoxu.

   Pokud se stejné testy opakují neustále dokola, neodhalí nakonec žádné nové defekty. Pro odhalení nových defektů může být nezbytné provést změny ve stávajících testech a testovacích datech, příp. je třeba vytvořit testy nové (testy již nejsou efektivní pro odhalování defektů, stejně jako pesticidy již po čase neúčinkují při ničení hmyzu). V některých případech (jako je například automatizované regresní testování) má však pesticidní paradox pozitivní přínos, kterým je relativně malý počet regresních defektů.

6. Testování je závislé na kontextu.

   Testování se provádí odlišně v různých kontextech. Například řídicí průmyslový bezpečnostně kritický software se testuje jinak než mobilní aplikace pro e-commerce. Dalším příkladem je testování v agilním projektu, kde testování probíhá odlišně od testování na projektu s využitím sekvenčního životního cyklu vývoje softwaru.

7. Nepřítomnost chyb je klam.

   Některé organizace očekávají, že testeři dokážou provést všechny možné testy a najít všechny možné defekty, ale Principy 1 a 2 nám říkají, že to možné není. Dalším omylem (tj. mylnou představou) je očekávání, že pouhým nalezením a odstraněním velkého počtu defektů lze zajistit úspěch systému. Například i přes důkladné testování všech specifikovaných požadavků a odstranění všech zjištěných defektů by mohl přesto vzniknout obtížně použitelný systém, který by nesplňoval potřeby a očekávání uživatelů, příp. by nepřinesl takovou (vyšší) hodnotu v porovnání s jinými konkurenčními systémy“ (1, s. 15-16).

3.5.3 Proces testování

Pro efektivní testování je nutné se řídit nejenom principy, ale také zvolit vhodnou strategii testování. Pro zachování a udržení přehledu v průběhu testování je vhodné si rozdělit testování do jednotlivých procesů. Procesy testování jsou zvoleny dle testovací sady daných činností, jelikož testování je ovlivněno velkým počtem faktorů, a z tohoto důvodu není možné sestavit na procesy testování univerzální návod. Pro optimální nastavení procesů je nutné provést analýzu faktorů, které testování ovlivňují.

Základní faktory ovlivňující procesy testování:

- Model životního cyklu vývoje softwaru, společně se zvolenou metodikou projektu
- Oblast podnikání
- Politika a procesy společnosti
- Typy testů a jejich úroveň
- Projektová a produktová rizika
- Provozní aspekty:
    - Finanční rozpočet
    - Časový plán
    - Požadavky zákazníka
    - Složitost projektu

Na základě analýzy faktorů dojde k sestavení jednotlivých činností a úkolů spojených s testováním. V další fázi testování dochází k vytvoření pracovního produktu, na jehož základě bude zpracována vzájemná trasovatelnost mezi pracovním produktem a testovací bází.

Činnosti a úkoly spojené s testováním se dělí do základních kategorií:

- Plánování testování: má na starosti management testovaní. Cílem těchto činností je definovat postupy, zvolit vhodné techniky testů na základě specifických požadavků a dále navrhnout časový harmonogram jednotlivých činností splňující dodržení termínů. Harmonogram a plány je možné v průběhu projektu upravovat na základě podnětů vycházejících z monitorování a řízení projektu.
- Řízení a monitoring testování: monitoring testování se zabývá průběžným porovnáním navrhovaného plánu s realizovanou skutečností prostřednictvím sledovacích metrik, které jsou uvedeny v plánech testování. Hlavním cílem řízení testování je splnění požadavků určených plánem testování na základě výstupních dat

poskytnutých z monitoringu testování. Primární činností řízení testování je řídit projekt tak, aby odchylka vykonaných činností byla co nejmenší oproti plánu. Pokud je odchylka příliš velká, je nutná komunikace s odpovědnými osobami prostřednictvím reportu. Na základě jejich rozhodnutí dojde k přepracování plánů nebo k úpravě testování, aby byla zachována potřebná úroveň kvality testování dané komponenty či produktu.

- Testovací analýza: se zabývá analýzou testovací báze podle vlastností, které je nutné splnit pro jednotlivé testovatelné komponenty a zvolit vhodné testovací podmínky. Testovací analýza se rozděluje na následující hlavní činnosti:
    - Analýza testovací báze:
        - Specifikace požadavků: identifikuje funkcionální a systémové požadavky. Dále se zabývá popisem uživatelských scénářů a případů užití testované komponenty nebo systémy, které se dále dělí dle chování na funkcionální nebo nefunkcionální
        - Informace o návrhu a implementaci: jedná se o potřebnou dokumentaci, která obsahuje specifikace rozhraní a návrhové, grafy, diagramy modelování, architektury systému a v poslední řadě popisují strukturu jednotlivých komponent nebo celého systému
        - Implementace: se zaměřuje již na samostatný kód, dotazy metadata, dále pak na databáze a rozhraní jednotlivých komponent nebo systému
        - Reporty o analýze rizik: popisují funkcionální, nefunkcionální a strukturální aspekty jednotlivých komponent nebo systému
        - „Hodnocená testovací báze a položek s cílem identifikace různých defektů se dále dělí na: nejednoznačnost, opomenutí, nekonzistence, nepřesnosti, rozpory, nadbytečnosti
    - Identifikace vlastností a sad vlastností určených pro testování
    - Definice a stanovení priorit testovacích podmínek pro každou vlastnost na základě analýzy testovací báze a zohlednění funkcionálních, nefunkcionálních a strukturálních charakteristik, dalších byznysových a technických faktorů a úrovně rizik
    - Zachycení obousměrné trasovatelnosti mezi každou položkou testovací báze a souvisejícími testovacími podmínkami“ (1, s. 18-19).

- Návrh testů: spočívá ve vytvoření testovacích případů a jejich sad vycházející z testovaných podmínek. Na základě zpracování návrhu testů je popsáno, jak nejvhodněji má být software otestován. K tomu je nutné použít správné druhy metodik, které splní požadavky na otestování určitých podmínek. Při správně zvolené metodice je možná identifikace defektu již v průběhu návrhu testů ještě před samotným otestování, což zaručuje rychlejší a méně nákladnou opravu.
- K hlavním činnostem návrhu testů se řadí:
    - Návrh a stanovení priorit testovacích případů a jejich sad,
    - Identifikace potřebných podpůrných testovacích dat pro testovací podmínky a testovací případy,
    - Návrh testovacího prostředí a identifikace potřebné infrastruktury a nástrojů,
    - Zachycení obousměrné trasovatelnosti mezi testovací bází, testovacími podmínkami a testovacími případy
- Implementace testů: je poslední činností před provedením samostatných testů. Spočívá k vytvoření testovacího softwaru, který je nezbytný k realizaci testů, který obsahuje jednotlivé testovací případy seřazené do posloupnosti, jak mají být prováděny. Implementace a dokumentace může být zároveň i realizací testů, které se využívá při průzkumném testování a také při testovaní založené na zkušenostech.
  „Implementace testování se rozděluje do hlavních kategorií, které jsou:
    - Definice a prioritizace testovacích procedur: obsahuje také realizaci automatizovaných testovacích skriptů
    - Vytvoření testovacích sad z testovacích procedur: obsahuje také realizaci automatizovaných testovacích skriptů
    - Zařazení testovacích sad v rámci harmonogramu provádění testů: je provedeno tak, aby bylo docíleno efektivního provedení testů
    - Vytvoření testovacího prostředí: obsahuje vhodné a správně nastavené nástroje potřebné k vytvoření testovacího prostředí sady testovacího vybavení, virtualizace služeb, simulátorů včetně položek infrastruktury
    - Příprava testovacích dat a zajištění jejich správné integrace do testovacího prostředí,
    - Ověření a aktualizace obousměrné trasovatelnosti: spočívá k ověření vzájemného propojení mezi testovací bází, jednotlivými testovacími

podmínkami, testovacími případy, testovacími procedurami a testovacími sadami.

- Realizace testů: obsahuje činnosti související s prováděním testů, které se vykonávají na základě předem vytvořeného harmonogramu.

  Hlavní činnosti, které se zabývají prováděním testů:

  - Zaznamenávání identifikátorů a verzí jednotlivých položek testování nebo testovaných objektů, testovacích nástrojů a testwaru,
  - Provádění testů buď ručně nebo pomocí nástrojů pro provedení testů,
  - Porovnávání skutečných a očekávaných výsledků,
  - Analýza anomálií pro určení jejich pravděpodobných příčin,
  - Hlášení defektů na základě pozorovaných selhání,
  - Zaznamenávání výsledků provedení testů,
  - Opakování testovacích činností, a to buď jako důsledek akce vykonané v souvislosti s nějakou anomálií nebo jako součást plánovaného testování,
  - Ověření a aktualizace obousměrné trasovatelnosti mezi testovací bází, testovacími podmínkami, testovacími případy, testovacími procedurami a výsledky testů“ (1, s. 20-21).

Dokončení testování: obsahuje činnosti, které se zabývají sběrem výstupních dat z provedeného testování za účelem dokončení projektu.

„Hlavní činnosti, které souvisejí s dokončením testování:

- Kontrola, zda jsou všechny reporty o defektech uzavřeny, zadání změnových požadavků nebo položek produktového backlogu vztažených k zadání defektů, které nebudou řešeny,
- Vytvoření souhrnného reportu z testování, který by měl být komunikován zainteresovaným stranám,
- Dokončení a archivace testovacího prostředí, testovacích dat, testovací infrastruktury a dalšího testwaru pro pozdější opakované použití,
- Předání testwaru týmům, které mají na starost údržbu, případně i dalším projektovým týmům nebo jiným zainteresovaným stranám, které by ho mohly dále využít,
- Analýza ponaučení získaných z dokončených testovacích činností s cílem stanovit potřebné změny pro budoucí iterace, vydání a projekty,
- Použití získaných informací s cílem zlepšit zralost procesu testování“ (1, s. 21).

3.5.4 Úrovně testování

Úrovně testování jsou základní testovací činnosti, které vychází z procesu testování. Jednotlivé činnosti, které jsou součástí plánovaní na základě životního cyklu vývoje modelu se provádí na daných úrovní v průběhu celého vývoje softwaru od prvotního testování jednotlivých komponent až po kompletní testování celých systémů. Specifikační cíle, testovací případ, testovací objekt, testovací báze dále pak specifikace odpovědnosti, identifikace a popis defektů patří ke společným vlastnostem, které jsou využívány ve všech úrovní testování.

Úrovně testování se dělí na:

- Komponentní testování
- Integrační testování
- Systémové testování
- Akceptační testování

KOMPONENTNÍ TESTOVÁNÍ se využívá pro samostatné testování jednotlivých komponent. Z pravidla testování probíhá izolovaně od zbytku systému a zaměřuje se na testování funkcionalit, která spočívá např. k ověření správnosti výpočtů, dále se provádí strukturální testování, jehož smyslem bývá většinou testování rozhodnutí. Následně se zabývá nefunkcionálním testováním, do kterého patří např. hledání úniku v paměti. Většinou se pro komponentní testování využívají manuální testy. V některých metodikách, kde dochází k průběžným změnám v kódu, zejména využívající inkrementálně – iterativní modely se používají automatické regresivní testy, kdy se ověřuje, zda provedené změny neměly negativní vliv na změnu funkčnosti. Pro testování komponent je nezbytný přístup k testovanému kódu, proto nejčastěji provádí testování vývojář, který kód implementoval. Po implementaci kódu vývojář rovnou navrhne a provede testy dané komponenty. Výhody komponentního testování spočívají v precizním ověření funkčnosti dané komponenty, dále pak včasném odhalení defektů, které jsou ihned opraveny bez zdlouhavého managementu reportování defektů. Tím dochází ke zvýšení kvality komponenty a je docíleno snížení rizika proniknutí defektů do vyšších úrovní testování. Při vývoji rozsáhlých systémů může komponentní testování na první pohled připadat až nevýhodné, z časového hlediska a finanční náročnosti. To je závislé na zvážení managementu testování, zda produkt vyžaduje tak důkladné testování.

INTEGRAČNÍ TESTOVÁNÍ se rozděluje do dvou různých úrovní podle charakteru testování na integrační testování komponent a systémové integrační testování. V případě

modelů, kde dochází k průběžným změnám v kódu se podobně jako u testování komponent využívají automatické regresní testy, kterými lze ověřit nenarušení funkčnosti zavedením nových změn. Využívá se způsob funkcionálního, nefunkcionálního a strukturálního testování. Testovací bází integračního testování mohou být např. návrhy softwaru a systémů, sekvenční diagramy, specifikace rozhraní, komunikační protokoly a další. Testovanými objekty zpravidla bývají subsystémy, databáze, infrastruktura, aplikační rozhraní a různé mikroslužby.

- Integrační testování komponent se používá pro ověření vlastní integrace. Zaměřuje se na vzájemnou komunikaci mezi jednotlivými moduly, nezabývá se vnitřní funkcionalitou jednotlivých komponent, která je předmětem komponentního testování. Obvykle je testování prováděno automatickými testy. Testování provádějí většinou samotní vývojáři. Odhalené defekty, které jsou typické pro integrační testování komponent jsou nesprávná nebo chybějící či nesprávně kódovaná data, nekompatibilita rozhraní, výpadky komunikace mezi komponentami a další.
- Systémové integrační testování se zabývá interakcí a rozhraním mezi systémy, službami nebo balíky. Systémové integrační testování lze využít i pro testování komunikace s externími zdroji, rozhraními nebo organizacemi. Podobně jako u integračního testování komponent se systémové integrační testování zaměřuje pouze na komunikaci mezi systémy, službami nebo rozhraními. Není cílem ověřovat jejich vnitřní funkčnost. Testováním se zabývají především testeři, u kterých se očekává znalost systémové architektury. Typické defekty odhaleny systémovým integračním testováním jsou nesprávná nebo chybějící či nesprávně kódovaná data, nesoulad rozhraní, ztráta komunikace mezi systémy, nejednotná struktura předávání zpráv mezi systémy a další. Cílem integračního testování je nalézt defekty na základě funkcionálního nebo nefunkcionálního způsobu testování, které se mohou vyskytovat v interakci mezi komponentami, systémy či rozhraními. Dochází tímto způsobem ke snížení rizika proniknutí defektů do vyšších úrovní testování a zároveň ke zvýšení kvality výsledného produktu. V případě testování většího počtu komponent nebo rozsáhlých systémů nastává zvýšení rizika přehlednutí defektů a delší časová náročnost. Právě pro tyto případy je doporučeno nasadit průběžnou integraci, která většinou zahrnuje několika úrovňové automatizované regresní testování.

SYSTÉMOVÉ TESTOVÁNÍ se zaměřuje na testování chování celého systému nebo produktu jako celku, jak po stránce funkcionálního, tak nefunkcionálního chování. Systémové testování také jako testování komponent a integrační testování využívá pro testování automatické regresní testy, kterými je docíleno ověření, že v průběhu zavedení změn nebyly narušeny funkčnosti nebo schopnosti end-to-end činnosti systému. Testovací báze v systémovém testování jsou specifikace systémových a softwarových požadavků, reportů z analýzy rizik, případy užití společně s uživatelskými scénáři, stavové diagramy a dále systémové a uživatelské příručky. Testované objekty zpravidla bývají aplikace, hardwarové, softwarové a operační systémy a dále konfigurace systémů s konfiguračními daty.

Mezi obvykle odhalené defekty a selhání systémovým testováním jsou nesprávné výpočty, nesprávné a neočekávatelné chování systému po funkcionální nebo nefunkcionální stránce, nesprávné fungování prostředí systému, chybné zpracování end-to-end funkcionálních úloh a dále rozdíly ve fungování systému oproti systémové a uživatelské dokumentaci. Systémové testování je prováděno většinou nezávislými testery na základě specifikace systému nebo produktu. Pro testování jsou klíčové zvolené vhodné techniky, které nejvhodněji otestují funkčnost systému. Jednou z těchto technik je použití rozhodovací tabulky pro ověření chování systému dle specifikace. Testování tímto způsobem odhaluje defekty také ve specifikacích jako je absence uživatelských scénářů a nevhodně definované požadavky, což může vést k falešně-pozitivním nebo k pozitivně-falešným výsledkům a je snížena účinnost nalezení závažných defektů. Ke snížení vzniku těchto situací je důležité včasné zapojení testerů již do statického testování a také zpřesňování uživatelských scénářů.

AKCEPTAČNÍ TESTOVÁNÍ se zabývá stejně jako systémové testování funkčností a chováním celého systému před nasazením systému do provozu k používání zákazníkem. Jedná se již o poslední fázi testování, u kterého není hlavním cílem odhalení defektů, ale ověření kvality systému jako celku, zda je systém kompletní a splňuje funkce dle specifikace. Při odhalení velkého množství defektů v průběhu akceptačního testování vzniká závažné projektové riziko, které může mít pro systém či projekt negativní důsledky. Důsledkem je nedostatečné testování v raných fází projektu vývoje softwaru. Testovací báze pracovních produktů pro akceptační testování se používají např. byznysové procesy, uživatelské nebo byznysové požadavky, dále regulatorní normy, smlouvy a nařízení, systémové požadavky, instalační postupy, systémové a uživatelské dokumentace. Objekty testování patřící do akceptačního testování jsou např. testovaný systém, konfigurace systému

a konfigurační data, byznysové procesy pro plně integrované systémy, procesy provozu a údržby, formuláře, reporty, stávající a konvertovaná výrobní data. Pro akceptační testování jsou obvyklé defekty např. systémové pracovní toky nesplňují byznysové požadavky nebo uživatelské požadavky, nesprávná implementace byznysových pravidel, nesplněné smluvní nebo regulatorní požadavky, nefunkcionální selhání, vysoká bezpečnostní zranitelnost systému, nedostatečná výkonnost systému pod zátěží nebo nesprávná funkčnost na podporované platformě.

Akceptační testování se rozděluje na:

- Uživatelské akceptační testování (UAT): se zabývá ověřováním systému z uživatelského pohledu. Testování se provádí již v reálném nebo simulovaném prostředí organizace a ověřuje se, zda systém splňuje potřeby a požadavky uživatele a také zda vykonává funkční procesy na základě požadavků s minimálními riziky narušení podnikových procesů.
- Provozní akceptační testování (OAT): ověřuje se v simulovaném prostředí správnost provozních faktorů, do kterých se zahrnuje otestování zálohování a obnovování ze záloh, provádí se testy instalace, odinstalace, upgrade sytému, obnova systému po havárii a správa uživatelů, dochází k testování údržby systému, úloh načítání, migrace dat, ověřuje se zranitelnost a zabezpečení systému a dále se provádí testování výkonnosti systému. Tyto testovací činnosti vykonávají provozní zaměstnanci nebo správci systému. Smyslem provozních akceptačních testů je ověření, že zaměstnanci budou schopni zajistit správné chování a obsluhování systému v běžných provozních i ve výjimečných podmínkách.
- Smluvní a regulatorní akceptační testování: se zaměřuje na prohloubení důvěry jejíž cílem je splnění smluvních a regulatorních požadavků. Smluvní akceptační testování se provádí pro ověření smluvních akceptačních kritérií, které by obvykle měly být určeny již v době uzavíraní a odsouhlasení smlouvy. Provádí je většinou nezávislý testeři nebo uživatelé.
- Dalším typem testování je regulatorní testování, které se zaměřuje na kontrolu celé řady předpisů, které by měly být v souladu s vládními, právními nebo bezpečnostními předpisy. Testování provádějí nezávislí uživatelé nebo testeři, v některých případech mohou být přítomni zástupci regulatorních orgánů.

ALFA A BETA TESTOVÁNÍ se používá pro získání zpětné vazby komerčního krabicového softwaru prostřednictvím současných zákazníků, nových potencionálních

zákazníků a také od provozovatelů systémů ještě před uvedením softwaru na trh. Alfa a beta testování se od sebe odlišuje v prostředí, ve kterém se testování realizuje. Alfa testování provádí současní nebo potencionální zákazníci a provozovatelé systému nebo nezávislý tým přímo ve vývojářském pracovišti. Beta testování je také realizováno potenciálními nebo stávajícími zákazníky nebo provozovateli systému. Odlišuje se tím, že se již neprovádí na vývojářském pracovišti, ale uživatelé testují na svých vlastních pracovištích a zařízení. Obvykle alfa testování předchází beta testování, ale nemusí to být pravidlo. Beta testování může být realizováno i bez předchozího alfa testování. Alfa a beta testování se využívá pro zvýšení důvěry produktu u stávajících nebo nových zákazníků a také provozovatelů v běžném každodenním používáním ve svém pracovním prostředí. Dalším cílem je získat zpětnou vazbu při používání systému a popřípadě odhalení defektů, které nemohly být objeveny z důvodu absence simulace podmínek a prostředí, ve kterých je systém používán (23).

3.5.5 Psychologie testování

Na první pohled by se mohlo zdát, že psychologie nepatří do oblasti testování. Tato domněnka je velký omyl. Na vývoji softwaru se velkou měrou podílí lidská energie a s ní je spojená lidská psychika, kterou testování ovlivňuje pozitivně, ale i negativně. V průběhu statického testování, kdy dochází k revidování požadavků od zadavatele a upřesňování uživatelských scénářů. Také v rámci dynamického testování dochází k odhalování defektů. Odhalení defektů působí negativně na psychiku softwarových vývojářů, jelikož se domnívají, že jejich kód je správný a neobsahuje chyby. To může způsobit, že osoba může záporně přijmout negativní informace, které jsou v rozporu s jeho názory. Tento stav se nazývá konfirmační zkreslení a jedná se o jeden z faktorů psychologie osobnosti. Dalším faktorem, se kterým je možné se setkat je kognitivní zkreslení, které má za následek negativní přijmutí a pochopení informace na základě provedeného testování. Dále běžným faktorem, který se vyskytuje v oblasti testování je obviňování osob, které defekty objevili, a proto někdy dochází k tomu, že testování je vnímáno jako destruktivní činnost místo toho, aby byla považována za aktivitu, která pomáhá zvyšovat kvalitu produktu a jeho úspěšné dokončení. Proto je velice žádoucí snížit napětí, které může vzniknout mezi jednotlivými členy týmu jakou jsou testeři, analytici, vlastníci produktu, návrháři a vývojáři. Ke snížení napětí mezi členy týmu by měla přispět konstruktivní komunikace o selhání, chybách a defektech. Problém je způsoben tím, že vývojáři a testeři přemýšlejí odlišně.

Hlavní myšlenkou vývojáře je vytvořit a navrhnout dokonalý a bezchybný produkt. Zabývají se hlavně návrhem a tvorbou řešení, než aby přemýšleli nad věcmi, které by nemusely fungovat správně, a proto je pro ně těžké si připustit chyby ve vlastní práci. Oproti tomu hlavní myšlenka testera je ověřit validaci produktu a nalézt defekty. Jeho myšlení obsahuje jistý kritický pohled, zvídavost a profesionální pesimismus. Rozdíly v myšlení nemusí přinášet jen spory, ale naopak spojením rozdílů lze docílit vytvoření produktu vysoké kvality. Dále by měla být upřednostňována snaha o spolupráci mezi členy týmu a nepodporovat rivalitu. Vyzdvihnout jejich společné cíle, kterými vznikne kvalitní produkt. Na zdokonalování svých mezilidských vztahů se musí podílet nejen manažeři testování, ale také testeři vhodným předáváním informací z testování o nalezení defektů či chyb. Celkový přístup mezi lidskými vztahy v týmu je závislý na organizaci, který vychází z modelu životního cyklu vývoje softwaru, protože v některých týmech vývojáři testují svůj kód sami, ale jinde zase kód testují nezávislý testeři (24).

3.6 Techniky testování

Předchozí kapitoly se zabývaly především teorií před samostatným začátkem testování. Cílem této kapitoly bude již přiblížení jednotlivých technik používaných pro testování softwaru.

3.6.1 Typy testů

V této kapitole budou vysvětleny druhy technik testování v závislosti na typech testů, do kterých je zařazeno funkcionální, nefunkcionální testování a testování související se změnami. K testování se dále ještě používají techniky testování černé a bíle skříňky a testování založené na zkušenostech, což bude předmětem následujících kapitol. Typy testů se rozlišují podle toho, jakým způsobem a jaká část softwaru se má otestovat. Cílem testů je obdržet objektivní výsledek. To je možné docílit nasazením vhodných testů podle konkrétní charakteristiky testování. Typy testů se dělí na základě charakteristiky podle toho, zda vyhodnocují funkcionální nebo nefunkcionální chování. Při testování funkcionálního a nefunkcionálního chování jsou výstupem výsledné vlastnosti, které charakterizují kompletnost, spolehlivost, vhodnost, správnost, výkonnost, použitelnost a kompatibilitu (25).

FUNKCIONÁLNÍ TESTOVÁNÍ, jak je již z názvu zřejmé, se zabývá testováním funkčnosti systému. Interpretuje to, „co by měl systém správně dělat“. Testy musí být

vytvořené tak, aby byly schopné ověřit funkčnost systému na základě funkčních požadavků, které jsou obsaženy v potřebných dokumentech jako jsou např. uživatelské scénáře, případy užití nebo funkční specifikace. Pro důsledné ověření funkčnosti systému je nutné provést funkcionální testování nejlépe ve všech úrovních od komponentního testování až po akceptační testování. Testy nejčastěji bývají řešeny technikou černé skřínky pro ověření funkcionality samotných komponent nebo i celých systémů. Zaměření testů v jednotlivých úrovní se od sebe odlišuje, ale zásadní pro důsledné ověření je rozsah funkcionálního pokrytí vytvořenými testy. Funkcionální testování musí navrhovat a provádět osoby, které ovládají potřebné dovednosti a znalosti v oboru pro který je daný software vytvořený.

NEFUNKCIONÁLNÍ TESTOVÁNÍ má za cíl vyhodnotit software či systém z pohledu použitelnosti, výkonnosti nebo bezpečnosti. Interpretuje to, „jak dobře by se měl systém chovat“. Nefunkcionálnímu testování by měl být prověřen software nebo systém již v ranném stádiu vývoje softwaru, aby byly ověřeny charakteristické vlastnosti, co možná nejdříve a případné defekty byly odstraněny již v prvopočátku. V případě pozdního odhalení většího množství defektů nefunkcionálního charakteru může mít pro budoucí vývoj projektu až ohrožující důsledky. Nevyplývá z toho, že by software nebo systém neměl být podroben nefunkcionálnímu testování ve všech úrovních testování; jde o to, aby byl otestován hned v prvotní fázi vývoje. Podobně jako u funkcionálního testování, tak se obvykle využívá technika černé skřínky pro definování podmínek také u nefunkcionálního testování, které jsou extrémní zátěže pro testování výkonnosti nebo k analýze hraničních hodnot. Pro ověření, že je software či systém důkladně otestován se používá pojem pokrytí, který je uváděn v procentech a udává nakolik procent je testovaná část pokryta testy. Testy k nefunkcionálnímu testování musí navrhovat a provádět osoby, které disponují potřebnými znalostmi a dovednostmi související s danou problematikou jako jsou technologie a znalost slabých stránek bezpečnosti (26).

TESTOVÁNÍ SOUVISEJÍCÍ SE ZMĚNAMI se používá v případě zavedení změn v systému nebo softwaru na základě opravy nějaké chyby, zavedením nové nebo změněné funkcionality. Cílem tohoto způsobu testování je ověřit, zda skutečně došlo v případě odhaleného defektu k jeho odstranění a nedošlo opravou chyby ke vzniku jiného defektu. V případě implementace nové nebo změněné funkcionality se ověřuje, zda implementace nezpůsobila nežádoucí následky. Pro testování související se změnami se využívají dvě techniky, které jsou:

- Konfirmační testování se provádí za účelem ověření, zda provedené změny skutečně odstranily defekt. Software se otestuje jak testy, které selhaly tak je nutné vytvořit nové testy, které ověřují, zda defekty byly opravdu odstraněny.
- Regresní testování ověřuje, zda zavedené úpravy kódu ať už z důvodu opravy defektu či zavedením nové nebo změněné funkčnosti nedošlo k narušení chování komponent nebo celého systému. Regresní testy se všeobecně mění pomalu a pouštějí mnohokrát pro určení, zda nebyly do softwaru zavedeny vedlejší účinky, a proto jsou velice často automatizovány.

Konfirmační a regresní testování se doporučuje provádět na všech úrovních testování (27).

3.6.2 Statické testování

Statické testování je metodika testování softwaru, která pro své výsledky nevyžaduje spuštění softwaru na rozdíl od metodiky dynamického testování. Hlavní činností tohoto způsobu testování je revize zdrojového kódu a jednotlivých dokumentů k testovanému softwaru. Z počátku se jeví statické testování jako nepodstatné, ale ve skutečnosti by testování dokumentace nemělo být podceňováno. Jak již bylo zmíněno v předešlých kapitolách, čím dříve jsou defekty či chyby odhaleny tím je jejich odstranění snadnější; a je malá šance, že by došlo k ohrožení celého projektu. Z tohoto důvodu je klíčové začít software testovat co možná nejdříve formou statického testování. Statické testování se dělí do dvou kategorií testování dokumentace a statické testování kódu.

Testování dokumentace se označuje také jako revize. Jedná se o dokumenty, které jsou součástí softwaru jako jsou manažerské výkazy, uživatelské a technické dokumenty. Testování podléhají všechny druhy dokumentace, které musí splňovat stejná kritéria pro testování. Smyslem dokumentace je sdělit čtenáři potřebné informace srozumitelnou formou. Testování dokumentace se zaměřuje na základní vlastnosti, které jsou klíčová pro správnost dokumentace.

„Hlavní vlastnosti:

- Úplnost: v dokumentu jsou obsaženy veškeré informace odpovídající charakteru dokumentu, případně jsou uvedeny odkazy na zdroje, kde lze získat potřebné znalosti. Čtenář si při čtení dokumentu nemusí nic domýšlet. Jedná se o kritérium, které lze těžko ověřit, jelikož autoři dokumentace neposkytují veškeré informace jako jsou např. zápisky ze schůzek. Tím vzniká neúplnost dokumentu a tím může být narušena jeho interpretace.

- Správnost: veškeré informace v dokumentu musí být správně, resp. musí být pravdivé. Čtenář musí dostat správné informace, aby se případně mohl na základě dokumentu správně rozhodnout.
- Relevantnost: informace v dokumentu se musí týkat cíle projektu a účelu dokumentu. Zbytečné informace otupí čtenářovu pozornost a může se stát, že přehlédne ty důležité. S relevantností informací v dokumentu nám může pomoci vhodná forma (šablona) dokumentu a její dodržování.
- Jednoznačnost: formulace musí být voleny tak, aby nebylo možné text interpretovat více způsoby. Chybou je také, pokud informace v dokumentu zamlžují čtenáři cíl projektu.
- Konzistence: text musí být konzistentní napříč celým dokumentem, ale i napříč celým projektem nelze v úvodní studii projektu deklarovat záměr, který je pak v dokumentaci návrhu softwaru popřen nebo změněn“ (2, s. 98).
- Jedná se o kritérium, které se obtížně posuzuje z důvodu velkého množství existující dokumentace nebo z vyššího počtu změn. Dále konzistentnost narušuje počet autorů v týmu a jejich různá interpretace a formulace stejných výrazů, která lze zvýšit vytvořením terminologického slovníků pojmů.

U testování dokumentace by mělo být v prvotní fázi ověřeno dodržení definovaných projektových šablon. V následující fázi by měl autor dokumentace provést korekci všech zásadních a hrubých chyb. Ve třetí fázi je doporučeno využít alespoň jednu z níže uvedených metod, jelikož po určitém čase není autor schopen odhalit svoje chyby a stává se vůči nim imunní.

„Doporučené metody k testování dokumentace:

- Neformální revize: znamená kontrolu jedním kolegou nebo několika málo kolegy autora dokumentu. Není nutné, aby vybraní kolegové byli v tvorbě podobného dokumentu zkušenější než autor. Výhodou je nízká formálnost kontroly a malé náklady na provedení kontroly. Nevýhodou je, že pokud nejsou kolegové dostatečně obeznámeni s projektem, pak není dostatečně otestována konzistence mezi dokumenty. Problém může nastat i tehdy, pokud je autor přítomen inspekci a poskytuje kolegovi vysvětlení a doplňující informace. Dokument by měl obstát sám o sobě.
- Walktrough: je setkání více účastníků, iniciované a vedené autorem dokumentu, které může být na různé úrovni formálnosti. Může se jednat o skupinovou kontrolu

výstupů, kdy autor představuje dokument ostatním, nebo o zkoušku prezentace či nácvik možných scénářů nanečisto. Výhodou je, že více kontrolorů má šanci najít více chyb. Nevýhodou je nutnost koordinovat skupinu kolegů.

- Technická revize: na rozdíl od předchozích metod vyžaduje určitou míru formálnosti. Na úrovni organizace by mělo být nastaveno, jak revize probíhá, jaké jsou její cíle a jaké jsou její výstupy. Výstupem by vždy měl být minimálně výstupní report, hodnotící kvalitu revidovaných dokumentů.
- Inspekce: je formální postup, kde dokument kontroluje celá skupina kontrolorů. Mělo by se jednat o kolegy, kteří mají dostatečné zkušenosti. Měli by být přítomni i lidé zodpovědní za jednotlivé oblasti projektu, kterých se dokument může týkat. Každý účastník inspekce by měl dostat dokument k posouzení předem. Na setkání věnovaném inspekci dokumentu jsou pak probírány již jen připomínky a otázky k dokumentu. Výhodou tohoto způsobu je, že nalezne velké množství chyb, na druhou stranu se ale jedná o velmi náročnou techniku z hlediska přípravy, organizace a časové náročnosti. Celý proces má na starosti moderátor, který není autorem posuzovaného dokumentu“ (2, s. 99).

Projekt, jehož zásadní úspěch je spojen s dokumentací, je vhodné pro ověření věrohodnosti dokumentu provést statické testování dokumentace na úrovních obsahu a formy. Pro ušetření času osoby, která provádí revizi je doporučeno zvolit postup, při kterém se nejprve identifikují všechny chyby v celém dokumentu a až poté se řeší jejich oprava.

STATICKÉ TESTOVÁNÍ KÓDU se označuje také jako statická analýza. U statického testování kódu se používají podobné postupy, které se využívají pro statické testování dokumentu s tím rozdílem, že ve statické analýze je možné využít více obecných metodik, podle kterých lze automaticky měřit a vyhodnocovat. Hlavním přínosem statického testování kódu je snaha vytvoření snadno udržovatelného, testovatelného a srozumitelného kódu pro programátora i testera. Uvedené vlastnosti se zároveň projevují i do dynamického testování, jelikož je kód opraven ještě před jeho samostatným spuštěním. Statické testování kódu pracuje s termínem pro specifikaci požadavků „kvalita zdrojového kódu“. Pro opravu defektů, chyb v kódu nebo přidání či úprava funkcionality je rozhodující právě kvalita kódu, která má za následek složitost změn. Statické testování kódu se zaměřuje na přesně definované postupy a formální náležitosti. Z těchto faktorů vychází analýza kódu, která patří k hlavním činnostem. Pro posouzení kvality kódu je nutné definovat konkrétní cíle, které je

potřeba splnit, jelikož nejde obecně definovat kvalita kódu pro různé programovací jazyky nebo zadané požadavky (28).

„Faktory, dle kterých se posuzuje kvalita:

- Udržitelnost a rozšiřitelnost: tento cíl se stanovuje, pokud se předpokládá, že celá aplikace se bude nějakou dobu muset udržet na trhu. Je pravděpodobné, že se objeví nějaké drobné chyby, které bude potřeba opravit, nebo bude nutné přidat novou funkcionalitu. Vznikající kód se bude v budoucnu rozvíjet a měnit. Nejspíš se bude měnit i tým lidí, kteří kód vytvořili. Proto je žádoucí mít kód dobře čitelný a srozumitelný, aby pozdější úprava kódu byla možná a nebylo levnější celý kód napsat znovu.
- Výkonová optimálnost: podmínky, v jakých se bude vznikající aplikace používat, vyžadují maximální rychlost nebo optimální práci s pamětí. V takovém případě se musí kód přizpůsobit těmto požadavkům, což může být někdy v rozporu s požadavkem srozumitelnosti a udržovatelnosti.
- Testovatelnost: při psaní kódu je nutné myslet na to, aby se kód „sám nebránil“ testování. Proto je třeba volit vhodnou architekturu aplikace a organizaci jednotlivých kusů kódu tak, aby se při testování usnadnila práci, nebo aby bylo testování vůbec proveditelné. Příkladem může být testování metody, která v sobě soustředí funkčnost z mnoha různých oblastí (2, s. 100).

„Při psaní kódu je doporučeno se vyvarovat obecným chybám, které mají za následek kódu nežádoucí jevy, do které jsou:

- Nevhodný návrh: vede k tomu, že kód není srozumitelný, nebo je těžké ho testovat například kvůli nemožnosti efektivně simulovat některé části kódu pro testovací účely.
- Nesoulad s rozhraními podle specifikace: může být zaviněn buď tak, že kód dělá něco jiného, než říká specifikace, nebo došlo k rozšíření rozhraní nad rámec specifikace. Jedná se o častý problém nesouladu specifikace a kódu, který vede k chybám v programu (nebo při komunikaci s programem).
- Chybné použití globálních proměnných: způsobuje chyby, u nichž je těžké identifikovat příčinu. Vyhledáním a odstraněním takovýchto chyb je nutné strávit mnoho času.

- Nedosažitelný kód: je takový kód, který nebude při běhu programu nikdy vykonán. Někdy je takový kód zřejmý, jindy ale může být problematické ho odhalit. Takový kód zbytečně způsobuje nepřehledný a zamlžující zbytek kódu.
- Inicializované, ale nevyužité proměnné nebo celé kusy nevyužitého kódu: většinou vzniknou při úpravách existujícího kódu, kdy proměnná nebo metoda už není potřeba, ale definice je stále přítomná v kódu. Chyba opět vede k vyšším nákladům na realizaci a údržbu testů, které již mohou být zbytečné“ (2, s. 100-101).

Pro statické testování kódu se používají kromě doporučení také metriky, které primárně neidentifikují příčinu nebo problém, který snižuje kvalitu kódu, ale spíše funguje jako nápověda pro posouzení kvality.

Metriky kvality kódu:

- Počet řádků kódu: jedná se o jednoduchou metodiku, která informuje o počtu řádků v kontextu celého kódu.
- Počet řádků na metodu: účelem této metriky je porovnání mezi různými částmi kódu napříč všemi třídami a metodami projektu.
- Cyklomatická složitost: definuje složitost programu na základě větvení programu. Čím je větvení rozsáhlejší tím se stává program méně přehledný. Dále roste počet kombinací, které je nutné otestovat a tím se testování stává náročnější a nákladnější.
- Provázanost tříd: určuje vzájemnou závislost tříd kódu mezi sebou. Vysoká provázanost kódu má za následek opět složitější testování.
- Pokrytí kódu testy: znázorňuje pokrytí jednotlivých řádků kódu testováním. Vyjadřuje se v procentech. Ideálně je mít kód 100% pokrytý, ale to není možné realizovat u všech testů. Pokrytí ovlivňují kritérii jako jsou např.: termíny, rozpočet, složitost kódu a jiné. Je nutné si položit otázku jaké procentuální pokrytí vyžaduje kód, aby byl dostatečně otestován a fungoval dle požadavků.

Ke statickému testování kódu se také dále využívají nejen metriky nebo různá doporučení, ale primárně metody, podle kterých je možné posoudit kvalitu kódu.

Metody statického testování kódu:

- Revize kódu: lze přirovnat k technické revizi dokumentu. Zkušenější programátor kontroluje, upozorňuje na chyby v kódu a vznáší dotazy. Revidovat by se měli jen malé části kódu, aby programátor, který provádí revizi měl dostatečný přehled o činnosti, kterou má kód vykonávat. Po provedení revize se kód vrací k přepracování nebo se již podrobuje dalšímu testování.

- Párové programování: je převzato z Extrémního programování, které se vyznačuje specifickým stylem programování. Kdy na stejném úkolu pracují současně dva programátoři na jednom počítači. Tento typ testování nevyžaduje přítomnost testera, jelikož programátoři se navzájem kontrolují, poskytují si vzájemně zpětnou vazbu a během vývoje odstraňují chyby a defekty.
- Automatické testování kódu: vychází z předpokladu strojového čtení kódu. Pro tento způsob testování se využívá programátorské vývojové prostředí IDE, které má na starosti dohled kódovacích standardů, které je možné nastavit dle interních pravidel. Automatické vyhodnocování a hlídání je možné využit i na ostatní metriky kódu (2, s. 97-103).

3.6.3 Testování černé skříňky

Testování černé skříňky také známé pod pojmem behaviorální testování nebo také jako black box, které je založené na vstupech a výstupech testovaného systému bez hlubších znalostí jeho vnitřní sktruktury. Využívá se pro funkcionální a nefunkcionální testování. Testování černé skřínky se využívá k nalezení chyb v operacích na vysoké úrovni jako jsou hlavní funkce, provozní profily a scénáře zákazníků. Hlavní činností je tvorba funkčních testů na základě toho, co by měl systém dělat nebo jak by se měl chovat na základě vložených vstupů. Testování černé skříňky také umožňuje vytvářet nefunkční testy založené na tom, jak by měl systém fungovat (29, s. 3). Testování černé skříňky vyžaduje podrobné pochopení toho jak produkt, pro který je systém vytvořený funguje, co je jeho hlavním cílem.
Pro testování černé skřínky se využívají následující techniky:

- Rozdělení tříd ekvivalence: jedná se o techniku při níž jsou testy realizovány na základě rozdělení dat na oddíly, aby bylo možné u všech hodnot určitého oddílu využít obdobný způsob zpracování. Ekvivalence tříd rozlišujeme na hodnoty, které jsou platné a neplatné. Platné hodnoty se nazývají „platná třída ekvivalence“ a jsou tehdy pokud je výsledek komponenty nebo systému akceptován jako validní. V opačném případě, pokud nejsou hodnoty systémem či komponentou akceptovány jedná se neplatné hodnoty, pro které se užívá název „neplatná třída ekvivalence“. Rozdělení tříd ekvivalence je možné testovat na všech úrovních.
- Analýza hraničních hodnot: je metoda, která vychází z techniky rozdělení tříd ekvivalence, kterou rozšiřuje o hledání minimální nebo maximální hodnoty. Tento

způsob testování lze využít pouze pokud jsou testovaná data seřazena a složena z numerických nebo sekvenčních hodnot.

Třídy ekvivalence rozdělujeme do to třech kategorií:

- Neplatná: hodnota je příliš nízká.
- Neplatná: hodnota je příliš vysoká.
- Platná: hodnota se nachází mezi hraničními hodnotami

Stejně jako u techniky rozdělení tříd ekvivalence je možné analýzu hraničních hodnot použít na všech úrovních testování.

- Testování dle rozhodovací tabulky: se využívá v případech, kdy systém pro svou implementaci využívá složitá pravidla. Prostřednictvím rozhodovací tabulky se určují jednotlivé vstupní podmínky na jejichž základě systém či komponenta provede výsledek, kterým jsou výstupní data. Pro vyhodnocení podmínek se nejčastěji využívají booleovské hodnoty pravdivé nebo nepravdivé. Minimální pokrytí pro techniku rozhodovací tabulky spočívá vytvoření alespoň jednoho testovacího případu pro každé rozhodovací pravidlo v tabulce. Pokrytí se uvádí v procentech a určuje poměr počtu testovaných rozhodovacích pravidel jedním testovacím případem k celkovému počtu všech rozhodovacích pravidel. Testování dle rozhodovací tabulky se provádí na všech úrovních testování.
- Testování přechodů stavů: je technika ověřující přechody stavů softwaru, komponenty nebo systému, které jsou závislé na určitých podmínkách nebo událostech. Pro tento způsob testování se používá diagram přechodů a tabulka přechodů. Diagram přechodů graficky znázorňuje pouze platné přechody a průchody jednotlivými stavy softwaru od vstupních stavů přes jednotlivé přechody závislé na určitých událostech a podmínkách až do koncových výstupních stavů. Tabulka přechodů na rozdíl od diagramů přechodů uvádí všechny platné i neplatné přechody mezi jednotlivými stavy i událostmi a výsledné stavy. Testy jsou vytvořeny tak, aby otestovaly jednotlivé posloupnosti stavů a prošly všechny platné i neplatné přechody. Technika testování přechodů je vhodná k použití zejména k testování vestavěného softwaru (embedded software).
- Testování případů užití: se zabývá testováním na základě popisu chování jednotlivých interakcí mezi aktéry a vznikajícím systémem. Aktér může být znázorněn osobou, systémem, hardwarem nebo komponentou. Pro testování případů užití je nutno popsat, jak má systém pracovat na základě reakce chování aktérů. Dále definovat vstupní podmínky, které vycházejí s interakcí a aktivit a na základě jejich splnění jsou obdržena

výstupní data. Pro vysvětlení se využívá grafického znázornění s využitím pracovních toků a diagramů, které jsou doplněny popisným jazykem neboli scénářem (30, s. 56-59).

V příloze A na obrázku 19 Schéma struktury testování černé skříňky je graficky znázorněn princip testování černé skřínky. Na levé straně vstupují do programu vstupní podmínky složené ze správných, hraničních a nesprávných hodnot, které jsou programem zpracovány, který je znázorněn černým obdélníkem. Z jeho pravé strany vycházejí výstupní hodnoty, které nabývají booleovských hodnot, pravda nebo nepravda.

Většině případů nezávislé testovací organizace se zaměřuji na behaviorální testování, protože z konkurenčních důvodů nedisponují podrobnou znalostí vnitřní struktury programu.

3.6.4 Testování bílé skříňky

Testování bílé skříňky, též na nazýváno jako strukturální testování nebo white box identifikuje chyby v nízkoúrovňových strukturálních prvcích. Jedná se o řádky kódu, databázová schémata, čipy, podsestavy a rozhraní. Testy pro testování bílé skříňky jsou navrženy na základě podrobné technické znalosti systému, které umožňují odhalit defekty ve vnitřní struktuře softwaru. Principem testování bíle skříňky je vytvořit strukturální testy, které testují samotný kód a datové struktury programu. Právě z toho důvodu se testování bílé skříňky využívá především ve stádiu vývoje systému (29, s. 2).

Podobně jako u testování černé skříňky tak i testování bílé skříňky využívá různé techniky k otestování, které jsou:

- Testování a pokrytí příkazů: udává počet spustitelných testů. Dále vyjadřuje procentuální pokrytí spustitelných testů na základě prověřených testů k poměru všech spustitelných testů v celém programu.
- Testování a pokrytí rozhodnutí: se zaměřuje na ověření rozhodování v kódu programu na základě výsledku, které vychází z rozhodnutí přímo v kódu. Dochází k prověření míst, kde se nachází rozhodovací příkazy. V tomto bodě se program dále větví dle vyhodnoceného rozhodnutí. Pokrytí je také vyjádřeno procentuálně a informuje o počtu výsledku rozhodnutí prověřených testy k celkovému počtu rozhodnutí v testovaném programu.

Techniky testování příkazů a rozhodnutí se od sebe odlišují ve svém významu pokrytí. Testování příkazu přináší nižší míru pokrytí, než je tomu u testování rozhodnutí.

Pokud je dosaženo 100% pokrytí příkazů znamená to, že všechny spustitelné příkazy v kódu byly otestovány alespoň jednou, ale není zajištěno otestování veškeré logiky rozhodování. V případě testování rozhodnutí, kdy je dosaženo 100% pokrytí, je splněno prověření veškeré logiky, jelikož testování ověřilo všechny výsledky rozhodnutí, a to i variant, u které neexistuje žádný provedený příkaz. Odhalit defekty v kódu lze za pomoci příkazů pokrytí, pokud nedošlo ke spuštěním jiným druhem testů. Pokud není možné běžnými testy ověřit rozhodnutí výsledků pravda nebo nepravda, je nutné k odhalení defektů použít testy, které pokrývají rozhodnutí. Pokud je dosaženo 100% pokrytí rozhodnutí je zaručeno 100% pokrytí příkazů, ale tato poučka neplatí v opačném pořadí (30, s. 59-60).
V příloze A na obrázku 20 Schéma struktury testování bílé skříňky je graficky znázorněn princip testování bílé skříňky. Pro její otestování jsou nezbytné znalosti vnitřních stavů, podmínek a rozhodnutí.

3.6.5 Testování založené na zkušenostech

Technika založená na zkušenostech vychází z dovedností, zkušeností a intuicí již zkušeného testera, který se již v minulosti setkal s testováním podobných systémů, softwaru nebo komponent. Jedná se o techniku, u které je obtížné měřit pokrytí nebo účinnost, jelikož vychází z přístupu zkušeností testera. Využívá se především pro definování testů, u které nelze snadno definovat běžnými systematickými technikami. Technika testování založená na zkušenostech využívá pro odhalení defektů tyto druhy technik:

- Odhadování chyb: je technika, která využívá získaných znalostí testera z minulých projektů. Tester dokáže odhadnout kde vývojáři nejčastěji dělají chyby, kdy dochází k selhání aplikací nebo kde se obvykle vyskytují defekty. Z takto získaných zkušeností tester nejčastěji vytvoří seznam chyb, defektů a selhání.
- Průzkumné testování: se používá v případech, kde je nutné zjistit podrobnější informace o komponentě nebo systému z důvodu nedostatečné nebo chybějící specifikace, ale také z časových důvodu. Tyto informace získává z výsledků testování. Průzkumné testování nemá předem definované testy a je závislé na zkušenosti a kreativitě testera. Techniku lze aplikovat jak k testování černé nebo bílé skříňky.
- Testování založené na kontrolních seznamech: vychází z předem vytvořených kontrolních seznamů testů splňující podmínky testování. Kontrolní seznamy vznikají na základě zkušenosti testerů, kteří dokáží odhadnout požadavky uživatele,

nejčastější místa výskytu defektů nebo slabá místa sytému způsobující selhání. Podle těchto seznamů dochází k návrhu, implementaci a provedení testů. Na základě provedené analýzy může být dále použit stávající kontrolní seznam anebo je možné provést aktualizaci stávajícího kontrolního seznamu a v nevyhovujícím případě je vytvořen úplně nový kontrolní seznam. Testování založené na kontrolních seznamech je možné použít pro funkcionální i nefunkcionální testování (30, s. 60-61).

3.7 Management testování

Nasazením sebedokonalejších metodik či procesů se nedosáhne úspěchu, pokud nebudou vhodně řízené, a právě z tohoto důvodu se tato kapitola bude zabývat řízením procesu testování. Hlavním principem managementu testování je zabezpečit a využívat zdroje, kterými jsou např. lidé, software, hardware, infrastruktura, k provedení procesů s přidanou hodnotou. Management je obecně úkolem pro managera. V oblasti testování je zodpovědný za procesy řízení test manager. Vzhledem k tomu, že testovací procesy přidávají hodnotu pouze tím, že přispívají k celkovému úspěchu projektu, který zabraňuje závažnějšímu selhání programu. Test manager musí odpovídajícím způsobem plánovat a řídit testovací procesy. Jinými slovy, test manager musí vhodně uspořádat testovací procesy, včetně souvisejících aktivit a pracovních produktů, podle potřeb a okolností ostatních zúčastněných stran, a všech aktivit, kterými je životní cyklus vývoje softwaru a jeho pracovní produkty. Jedná se např. o specifikaci požadavků zákazníka.
Struktura rolí managementu testování:

- Vývojáři, vedoucí vývoje a manažeři vývoje: mají na starost implementaci testovaného softwaru, dostávají výsledky z testování na jejichž základě musí podniknout kroky, např. opravit hlášené defekty.
- Databázoví architekti, systémoví architekti a designéři: jejich činností je navrhnout software, obdržet výsledky testů a na základě obdržených výsledků musí podniknout kroky, které povedou k realizaci řešení.
- Marketingoví a obchodní analytici: jejich úkolem je určit funkce a úrovně kvality softwaru. Často se také podílejí na definování potřebného pokrytí testů, revizi výsledků testů a rozhodování na základě výsledků testů.

- Vyšší management, produktoví manažeři a sponzoři projektů: jejich náplní práce je definice potřebného pokrytí testů, revize výsledků testů a rozhodování na základě výsledků testů.
- Projektoví manažeři: jsou odpovědni za řízení svých projektů k úspěchu, což vyžaduje vyvážení priorit kvality, harmonogramu, funkcí a rozpočtu. Často obstarávají zdroje potřebné pro testovací aktivity a spolupracují s Test Managerem při plánování a řízení testů.
- Technická podpora, zákaznická podpora a personál helpdesku: jejich úkolem je podpora uživatelů a zákazníků, kteří používají funkční a kvalitní software.
- Přímí a nepřímí uživatelé: používají software přímo nebo přijímají výstupy nebo služby vytvořené nebo podporované softwarem (31, s. 18).

3.7.1 Plánování testování

V oblasti plánování testování jsou využívány důležité pojmy, jejichž role je pro plánování nepostradatelná. Prvním pojmem je politika testování. Jedná se o dokument, který popisuje jednotlivé principy, přístupy a hlavní cíle testování pro určitou organizaci. Druhým pojem, který je rozhodující v plánování testování je strategie. Strategie obecně popisuje jednotlivé testovací úrovně a testy založené na životních modelech vývoje softwaru, dle kterých bude testování provedeno. Třetí pojem je přístup k testování, ve kterém dochází již k implementaci strategie testování v určitém projektu. Obsahuje rozhodnutí určená na základě cílů projektu, dále zahrnuje odhad rizik, metody návrhu jednotlivých testů, které musí splňovat vstupní kritéria. Čtvrtým pojmem je plán testování, který je výstupním procesem pro plánování jednoúrovňových testů. Jedná se o dokument, který obsahuje rozsah, přístup, zdroje a harmonogram testovacích aktivit. Dále jsou v něm uvedeny jednotlivé body nutné k testování, testovací prostředí, vstupní a výstupní kritéria jednotlivých úkolů testování a kdo je bude plnit. Posledním pátým pojmem je hlavní plán testování, který se liší od plánu testování v tom, že se využívá pro plánování více úrovňových testů (32, s. 40).

Strategie testování vysvětluje obecnou metodiku na jejichž základě je v organizaci realizováno testování. Ve strategii testování je zahrnuto, jakým způsobem se testování používá k řízení rizik projektů a produktů. Dále obsahuje rozdělení testování do úrovní a činností spojené s testováním na vysoké úrovni. Jedna organizace může používat různé

strategie pro různé situace, jejichž činnosti a procesy by měly splňovat zásady testování. Strategie se dělí do kategorií, které jsou:

- Analytická strategie: jedná se o analytický přístup, který vychází z testování rizik. Testovací tým analyzuje určité faktory pro testovací základ dle, kterých se stanoví testovací podmínky nutné k pokrytí testů.
- Strategie založená na modelu: je založená na tvorbě modelu testovacího prostředí, ve kterém by měl systém pracovat. Testovací prostředí obsahuje vstupy a podmínky, kterými je systém vystaven a zjišťuje se, jak se sytém v prostředí bude chovat. Modely mohou být vyvíjeny na základě hardwaru, softwaru nebo infrastruktury, které budou v běžném provozu požívány.
- Metodická strategie: je založena na charakteristikách, které jsou stanoveny dle souboru testovacích podmínek, jako jsou standardy kvality, kontrolní seznamy.
- Založené na shodě s procesem: jedná se o strategii, kdy se testovací tým řídí souborem procesů, které jsou definované výborem pro standardy nebo jinými odborníky, ve kterých je popsáno, jak by měla být řešena dokumentace, správná identifikace a použití testovací základny nebo organizace testovacího týmu.
- Reaktivní strategie: vychází z testování založené na defektech, kdy testovací tým vytváří návrh a implementaci testů, dokud není software přijat a může pracovat ve skutečném testovacím systému. Využívá se při průzkumném testování.
- Konzultativní strategie: jedná se o typ testování, které je řízené uživatelem, kdy testovací tým zpracovává vstupy od jedné nebo více klíčových stran, aby mohly být určeny testovací podmínky, které je nutno pokrýt.
- Strategie testování averze k regresi: jedná se o rozsáhlé automatické testování, při kterém je využívána technika k řízení regrese, hlavně automatizaci funkčních nebo nefunkčních regresních testů na jedné nebo více úrovních (33).

Výše uvedené strategie testování lze kombinovat mezi sebou tak, aby vhodně vyhovovaly konkrétním systémům a projektům. Na základě strategie je možné vytvořit hlavní plán testování, který obsahuje testovací činnosti, které mají být provedeny na daném projektu. Jsou v něm uvedeny konkrétní úrovně testování, které mají být provedeny a zároveň jsou popsány vztahy mezi danými úrovněmi, které odpovídají vývojovým aktivitám. Hlavní testovací plán lze přirovnat ke zjednodušenému Ganntovu diagramu, kdy z průsečíků jednotlivých požadavků nebo částí systému úrovní testování vytvoří aktivity. V hlavním plánu by mělo být uvedeno, jak testeři budou implementovat testovací strategii

pro daný projekt. Hlavní testovací plán by měl být vytvořen tak, aby splňoval politiku a strategii testování ve specifické oblasti. Pokud, ale tomu tak není, musí být v plánu vysvětleny odchylky a výjimky, které by mohly mít potencionální dopad. Struktura hlavního plánu testování se odlišuje dle politiky a strategie konkrétní organizace, ve které je vytvořen. Dále je také ovlivněn rozsáhlostí projektu. Hlavní plán testování by měl splňovat obecná doporučení, která by měla být splněna při plánování testování velkého projektu (31, s. 18-35).

„Body, které by měl obsahovat hlavní plán testování:

- záznam o schvalování s klientem,
- manažerské shrnutí,
- úvod
- formulace cílů a rozsahu testovacího projektu
- použitá dokumentace
- strategie testování
- přístup k testování
- organizace testovacího projektu
- infrastruktura pro testování
- řízení testovacího projektu:
- seznam rizik a opatření proti nim,
- odhady a projektový plán:
- použité termíny a zkratky“ (32, s. 40-41).

„Položky, které by měl obsahovat plán testování pro jednoúrovňové testování:

- Pro záznam o schvalování s klientem, manažerské shrnutí, cíl dokumentu, formulaci cílů a rozsahu testovacího projektu je doporučena prakticky stejná struktura jako v případě hlavního plánu testování.
- Následuje použitá dokumentace:
- Strategie testování
- Přístup k testování
- Infrastruktura pro testování
- Řízení testovacího projektu v dané úrovni testování
- Odhady a (projektový) plán“ (32, s. 42).

V průběhu testování dochází k pravidelné aktualizaci plánu testování dle toho, jak jsou činnosti ve skutečnosti realizovány.

3.7.2 Monitoring testování

Hlavní náplní monitorování testování je sběr potřebných informací, které umožňují zpětnou vazbu o testovacích činnostech. Shromážděné informace umožňují ověřit, zda testování pokrylo potřebné testovací případy. Shromážděnými informacemi lze získat přehled o provedené testovací činnosti jejíž výsledkem jsou získaná výstupní data, která by se měla splňovat definované hodnoty, které jsou uvedeny v plánu testování. Data potřebná pro monitorování testovaní lze získávat ručním nebo automatickým způsobem. Sběr informací se také využívá pro řízení testování, které na základě monitoringu z testování a pomocí potřebných metrik řídí činnosti, postupy či úkoly spojené s testováním softwaru v rámci životního cyklu vývoje softwaru. Hlavní činnosti spojené s řízením testování jsou zejména pravidelné změny v plánu testování na základě priority testů, identifikace rizik v průběhu projektu, či dostupnosti nebo nedostupnosti potřebných zdrojů a vhodného prostředí. S monitoringem testování jsou spojené metriky, které jsou potřebné ke shromáždění dat. Úkolem metrik je shromažďovat data v průběhu a v konečné fázi testování za účelem porovnávání realizovaného postupu oproti plánovanému harmonogramu a rozpočtu. Dále pak ověřit aktuální kvalitu testovaného objektu nebo posoudit, zda určené cíle jsou pokryty vhodným přístupem a efektem testování (34).

„Obecné metriky, které se využívají ke shromáždění informací:

- Procentní podíl práce, která již byla vykonaná při přípravě testovacích případů proti plánu,
- Procentní podíl práce, která byla již vykonaná při přípravě testovacího prostředí proti plánu,
- Postup v provádění testovacích případů,
- Informace o defektech, které obsahují např. hustotu defektů, zjištěné a opravené defekty, míra selhání a výsledky konfirmačních testů,
- Míra pokrytí požadavků, uživatelských scénářů, akceptačních kritérií, rizik nebo kódu,
- Dokončení úkolů, alokace a využití zdrojů a úsilí,
- Náklady na testování, včetně poměru nákladů a přínosů zjištění dalšího defektu nebo poměru nákladů a přínosů provedení dalšího testu“ (30, s. 70).

Shromážděné informace se využívají nejen pro monitoring a řízení, ale jsou potřebná k tvorbě reportů o testovací činnosti. V průběhu testovací činnosti se vytváří informativní dokument, který se označuje jako report o postupu prací. V závěru testování softwaru se vytvoří dokument, který se nazývá souhrnný report z testování. Reporty vznikají za účelem informovat zainteresované strany. V průběhu testování dodávají potřebné informace o postupu prací při testování a po dosažení výstupních kritérií vzniká dokument obsahující informace o výsledku testování. Tvorba reportů v průběhu monitorování a řízení testování je činností manažera testování (30, s. 70-71).

„Reporty o postupu testovacích prací by měly zahrnovat informace:

- stav jednotlivých testovacích činností a postup proti plánu testování,
- faktory, které brání postupu prací,
- shrnutí testování plánovaného pro další reportovací období,
- informace o kvalitě testovaného objektu.

Souhrnné reporty z testování by měly obsahovat informace:

- shrnutí provedených testů,
- informace o tom, co se dělo během testovacího období,
- odchylky od plánu, včetně odchylek v harmonogramu, trvání nebo pracnosti testovacích činností,
- stav testování a informace o kvalitě produktu s ohledem na výstupní kritéria nebo definici hotového,
- faktory, které zablokovaly nebo stále blokují další postup testování,
- metriky týkající se defektů, testovacích případů, pokrytí testů, postupu v činnostech a čerpání zdrojů,
- zbytková rizika,
- pracovní produkty z testování vhodné pro opětovné použití“ (30, s. 70-71).

3.7.3 Rizika testování

Riziko představuje událost, která v budoucnosti může způsobit negativní škody, jejíž úroveň je určena pravděpodobností, že událost vznikne a dále pak dopadem, pokud událost opravdu nastane. V oblasti vývoje softwaru se sledují produktová a projektová rizika. Produktové riziko se zabývá vznikajícím produktem, kterým je znázorněn jako specifikace, systém nebo komponenta. Znamená to, že výsledný produkt nemusí splňovat potřeby

uživatelů nebo zainteresovaných stran. Produktová rizika mohou být také spojena s charakteristickými vlastnostmi produktu.

„Uvádějí se tyto produktová rizik:

- Software nemusí plnit jeho zamýšlené funkce podle specifikace.
- Software nemusí plnit jeho zamýšlené funkce podle potřeb uživatelů, zákazníků nebo zúčastněných stran.
- Architektura systému nemusí plně vyhovovat některým nefunkcionálním požadavkům.
- Za určitých okolností může být nesprávně provedený konkrétní výpočet.
- Řídicí struktury smyčky mohou mít nesprávné kódování.
- Doba odezvy může být nedostatečná pro systém zpracování transakcí.
- Uživatelský prožitek nemusí naplňovat očekávání od produktu“ (30, s. 72).

Projektová rizika se netýkají přímo výsledného produktu, jak je tomu u projektového rizika, ale vznikem události, které mají za následek negativní ovlivnění zadaných cílů projektu. Projektová rizika lze rozdělit do kategorií dle zaměření na:

- Projektové problémy
- Organizační problémy
- Politicko-sociální problémy
- Technické problémy
- Problémy s dodavateli (35).

Z výše uvedených důvodů se provádí tzv. testování na základě rizik. Při testování na základě rizik se využívají techniky, které jsou formálně definovány jako odlehčené a těžké techniky. U odlehčených technik se sledují pouze dva faktory, které jsou pravděpodobnost a dopad. Požívají se jednoduché kvalitativní úsudky a měřítka. Zástupce lehké techniky je metodika např. PRAM (Pragmatic Risk Analyses and Management). Metodika PRAM pro určování rizika využívá škálu od 1 do 5, ve které se nejvyšší pravděpodobnost či dopad rozlišujeme dle stupnice 1 až 5, kdy hodnota 5 je nejnižší. Odlehčené techniky se vyznačují poskytnutím flexibility, jednoduchostí, použitelností a dostupností napříč týmy. Těžké techniky jsou daleko obsáhlejší, pracnější, kladou vyšší nároky na dokumentaci než lehké techniky. Používají se především v oblastech, které disponují vysokou míru rizik. Nejznámějším zástupcem v oblasti těžkých technik je FMA (Failure Mode and Effect

Analyses). Obvykle je nejdůležitějším faktorem k úspěchu testování založené na rizicích, zapojení týmu a zainteresovaných stran do identifikace a hodnocení rizik (32, s. 71-73).

3.7.4 Management defektů

Hlavním cílem testování je nalezení defektů. Takto nalezené defekty v průběhu testování musí být zaznamenávány, aby později mohlo dojít k jejich opravě. K tomu to účelu je do oblasti testování zahrnut management defektů, jehož úkolem je systematická správa a reportování nalezených defektů prostřednictvím nástrojů a metrik. V úvodu budou představeny základní pojmy a vysvětleny rozdíly mezi nimi, které jsou v oblasti managementu defektů důležité.

Základní pojmy:

- Chyba: je způsobena lidskou činností, jako je např. chyba v kódu.
- Defekt: je způsobený chybou, která má za následek neočekávaného chování softwaru oproti specifikaci. K odstranění defektu je nezbytné nejprve opravit chybu.
- Selhání: vznikne na základě defektu, které má za následek havárii části nebo celého systému.
- Incident: jedná se o situaci, která nastává v důsledku vady v produkčním prostředí a vyžaduje prozkoumání. Odstranění incidentu je mnohem nákladnější, než kdyby byl defekt odhalen v průběhu projektu.

V managementu defektů jsou přítomností defektů vystavena celá řada osob, které plní své projektové role. Osoba může zastupovat najednou i více projektových rolí.

Projektové role, které přijdou do styku s defekty:

- Tester: jedná se o roli, jejíž úkolem je ověřit správné chování systému či komponenty. V případě nalezení defektu tester posoudí a označí defekt příslušnými atributy a reportuje. Po opravě defektu tester opět ověří správné chování v místě, kde byl defekt objeven a také zjišťuje, zda opravou defektu nebyla ovlivněna funkce celého systému.
- Analytik defektů: jeho úkolem je zpracování defektů, které odhalil tester. Provede ověření, zda jsou defekty relevantní a nejsou duplicitní. Nejasné či nekompletní defekty jsou vráceny autorovi k jejich upřesnění nebo

k přepracování. Relevantní defekty jsou dále předány odpovědným projektovým rolím, které zajistí jejich opravu.

- Manažer testování: provádí kontrolu nad správou defektů. Zajišťuje pro analytiky eskalační autoritu v případech váznoucí komunikace. Informuje o průběhu testování, podává informace o defektech a jejich následných odstranění.
- Vývojář: cílem jeho práce je vymezit a provést opravu defektů. Podává informace o průběhu opravy defektů. V jeho kompetencích je vznést návrh k efektivnějšímu přetestování defektů regresním testováním.
- Projektový a produktový manažer: zajišťují prioritizaci oprav defektů společně s vývojářem a manažerem testování. Pro manažera testování je zároveň eskalační kontaktní osobou v důsledku komplikací spojených s opravou defektů (32, s. 149-150).

Defekty nemusí být způsobeny primárně chybou programátora v kódu programu, ale celou řadou jiných příčin, které nepřímo souvisejí s programováním.

„Obvyklé příčiny, které vedou ke vzniku potencionálních zdrojů defektů:

- Nesprávně pochopené potřeby uživatele vedoucí k chybné specifikaci požadavků.
- Nesprávně nebo neúplně definované požadavky.
- Požadavky nesprávně interpretované do funkčního designu.
- Nesprávně interpretovaný funkční design, a tím nesprávný technický design.
- Nesprávně pochopený technický design, a tím nesprávně naprogramovaný software.
- Nedostatečný výkon aplikace, špatná ovladatelnost, uživatelské nepohodlí (a další aspekty, které se týkají zejména nefunkčních požadavků).
- Nesprávně pochopené požadavky či funkční design, a tím defekty v samotných testovacích scénářích či skriptech“ (32, s. 150).

Pro podání potřebných informací ohledně defektu slouží tzv. report defektu, jehož cílem je informovat všechny zainteresované osoby o neočekávaném chování softwaru během testování za účelem sjednání jeho opravy. Dále také umožňuje manažerům testováním sledovat kvalitu vznikajícího produktu, která je závislá na odhalení množství defektů během testování. Report o defektech je možné využít pro zpětnou vazbu ke zlepšení procesů spojených s vývojem a testováním.

„Report o defektech by měl obsahovat níže uvedené informace:

- identifikátor,
- název a krátké shrnutí zjištěného defektu,
- datum reportu o defektu, autor a jeho organizační zařazení,
- identifikace položky testovaní (testovaná konfigurační položka) a prostředí,
- fáze životního cyklu vývoje, ve kterých byl defekt pozorován,
- popis defektu umožňující reprodukci a vyřešení, včetně protokolů, záloh databází, screenshotů nebo videozáznamů
- očekávané a skutečné výsledky,
- rozsah nebo míra dopadu (závažnosti) defektu na zájmy zúčastněných stran,
- naléhavost nebo priorita opravy,
- stav defektu,
- závěry, doporučení a záznamy o schvalování,
- globální problémy, například jiné oblasti, které mohou být ovlivněny změnou vyplývající z defektu,
- historie změn jako je posloupnost činností provedených členy projektového týmu s cílem izolace a opravy defektu a potvrzení korektnosti opravy,
- reference, včetně odkazu na testovací případ, který odhalil problém“ (30, s. 75).

Životním cyklem prochází nejenom vývoj softwaru či sytému, ale i defekt. Defekt se vyvíjí určitým životním cyklem, který se nazývá workflow. Defekt během životního cyklu prochází celou řadou fází, podle kterých je možné dohledat v jakém konkrétním stavu se defekt nachází. Pro jednotlivé role je znalost stavu defektu klíčová, protože na jejichž základě mohou vykonávat činnosti spojené s odstraněním defektu. Příklad jednotlivých fází defektu je vyobrazen v příloze A na 21 obrázku Schéma životního cyklu vývoje defektu. Doporučeným nástrojem pro řízení managementu defektů je komunikační mapa pro správu defektů. Úkolem komunikační mapy je znázornit jednotlivé komunikační kanály, které slouží pro přenos informací mezi jednotlivými zainteresovanými projektovými rolemi, zákazníky nebo zadavateli. Pokud se v organizaci využívá komunikační mapa, měli by ji znát všichni členové týmu, jelikož umožňuje zefektivnit komunikaci a tím také procesy spojené s projektem. Příklad komunikační matice je vyobrazen v příloze A na obrázku 22 Komunikační matice pro správu defektů v projektu. Dalším doporučeným nástrojem, který

je možno vyžít ve správě defektů je eskalační mapa, která se využívá především při obtížných a nestandardních situací způsobený eskalacemi v projektu. Hlavním úkolem eskalační mapy je definovat, kdo je v případě eskalace zodpovědná komunikační osoba pro danou oblast. V eskalační mapě nejsou uvedeny přímo konkrétní jména, ale pouze projektové role. Pokud je eskalační mapa vytvořena; je nutné, aby ji znali všichni členové týmu (32, s. 152-153). Management defektů je v každé organizaci řízen odlišně, jelikož se odvíjí od nastavení procesů, velikosti a politiky organizace. Jsou společnosti, které zaznamenávají a sledují defekty spíše neformálně. Všechny organizace by se měly snažit, o co nejvyšší efektivitu testování, která se vyznačuje co nejnižším počtem falešně-pozitivních defektů. Těmto defektům je možné předcházet už prvotní fází, ve které jsou správně a detailně definovány a pochopeny požadavky zadavatele nebo zákazníka.

4 Vlastní práce

Tato kapitola se již bude zabývat vlastní realizací testování softwaru. V úvodní části bude představena společnost a její procesy, které jsou využívány v rámci testování softwaru pro automatické převodovky do osobních automobilů. Dále budou popsány nástroje, hardware s příslušenstvím, které jsou určené k realizaci testování softwaru ve společnosti. V další části budou uvedeny procesy komponentního manuálního testování softwaru, které budou popsány v jednotlivých kapitolách a budou se zabývat procesy komponentního testování, přípravami specifikace testování, realizací testování a zpracování výsledků. Po manuálním testování bude následovat představení procesů automatického regresního testování. Na základě uvedených procesů bude provedena analýza jednotlivých procesů komponentního manuálního testování a automatického regresního testování. Dále budou podklady z jednotlivých analýz obou metodik testování po vyhodnocení použity k provedení návrhu optimalizace procesů manuálního komponentního a automatického regresního testování. Na základě optimalizace bude provedena implementace navrhovaného procesu, jehož cílem je snížit časovou náročnost na automatické regresní testování.

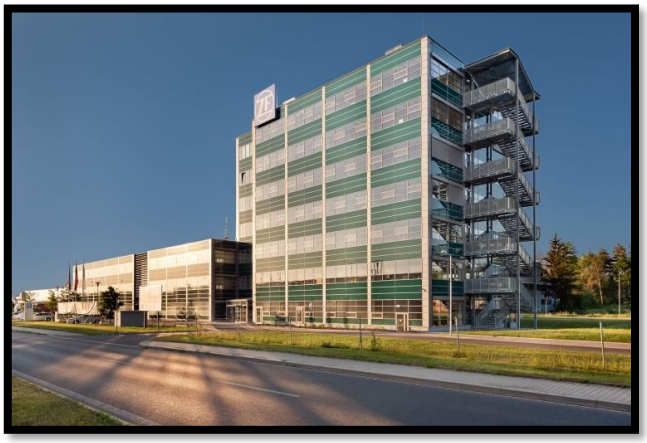
4.1 Představení společnosti

Společnost ZF Engineering Plzeň s.r.o. je jednou z deseti poboček, které zastupují globální koncern ZF Group v České republice. Historie vzniku koncernu ZF Group sahá až do roku 1915, kdy byla ve Fridrichshafenu založena společnost s ručením omezeným na výrobu ozubených kol, německy Zahnradfabrik a odtud pochází název ZF. Do současné doby se koncern ZF Group rozrostl na obrovskou společnost, která provozuje po celém světě 188 poboček a výrobních závodů v 31 zemích, ve kterých zaměstnává přibližně 157 500 lidí s obratem ve výši 38,3 mld EUR. Výrobní portfolio ZF Group se za dobu existence rozšířilo z pouhé výroby ozubených kol na společnost zabývající se vývojem a výrobou inteligentních systémů a pohonů pro osobní automobily, autobusy, nákladní, užitková a zemědělská vozidla celosvětově známých výrobců. Dále se zabývá výrobou a vývojem výrobků nacházející uplatnění v železniční, letecké a lodní dopravě. V současné době se ZF Group zabývá vývojem elektrifikace pohonů na různých typech vozidel a tím přispívá ke snižování emisí.

ZF Engineering Plzeň s.r.o. byla založena v roce 2007 a zaměstnávala pouze 50 lidí. V současné době je plzeňská pobočka složena ze třech pracovišť sídlících přímo v Plzni

s počtem 820 zaměstnanců. ZF Engineering Plzeň s.r.o. slouží jako vývojové centrum, které je součástí výroby pohonných ústrojí pro celou řadu dopravních prostředků jako jsou osobní automobily, nákladní a užitková vozidla, autobusy, zemědělské a stavební stroje, lodě a letadla. V současné době se zvyšuje podíl vývoje projektů v eMobilitě. Plzeňské vývojové centrum se specializuje na oblast elektroniky, softwaru, konstrukce, výpočtů, vývoje hardwaru a mechatroniky, a to jak na funkční, tak i systémové úrovni. Zároveň se zabývá vývojem různých SW nástrojů pro automatizaci a umělou inteligenci. V plzeňské pobočce, která je vyobrazena na obrázku 3 Budova plzeňské pobočky ZF Engineering Plzeň s.r.o. se nachází specializovaná pracoviště pro testování výkonové elektroniky, enviromentálního testování a speciální laboratoře.

Obrázek 3 Budova plzeňské pobočky ZF Engineering Plzeň s.r.o.



Zdroj: ZF Engineering Plzeň s.r.o.

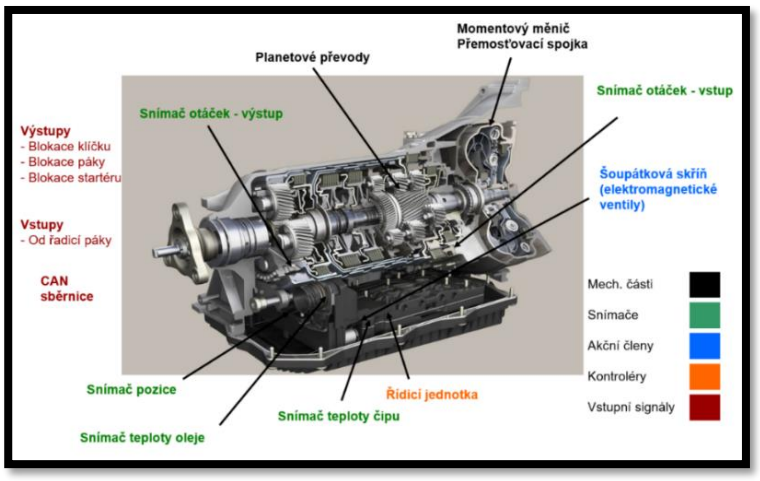
4.1.1 Popis automatické převodovky

V této kapitole bude představena automatická převodovka, nikoliv po mechanické stránce funkčnosti, ale především z pohledu softwarového řízení. Hlavní části automatické převodovky jsou znázorněny na obrázku 4 Schéma automatické převodovky. Do osobních automobilů jsou dodávány osmistupňové převodovky pro spalovací motory označené 8HP a 8HP50. Dále jsou dodávány osmistupňové převodovky pro hybridní pohon označené 8P a devítistupňové převodovky pro spalovací motory označené 9HP50. Automatické převodovky se kategorizují dle generace vývoje od první až po čtvrtou generaci.

Součástí automatické převodovky je řídící jednotka, která zajišťuje správnou funkci prostřednictvím softwaru, který zjišťuje logiku řazení nepřetržitým prováděním

matematických operací. Výsledky výpočtu se využívají k řízení akčních členů elektronického ovládání automatické převodovky. K nim se na prvním místě řadí elektromagnetické ventily umístěné v šoupátkové skříni.

Obrázek 4 Schéma automatické převodovky



Zdroj: ZF Engineering Plzeň s.r.o.

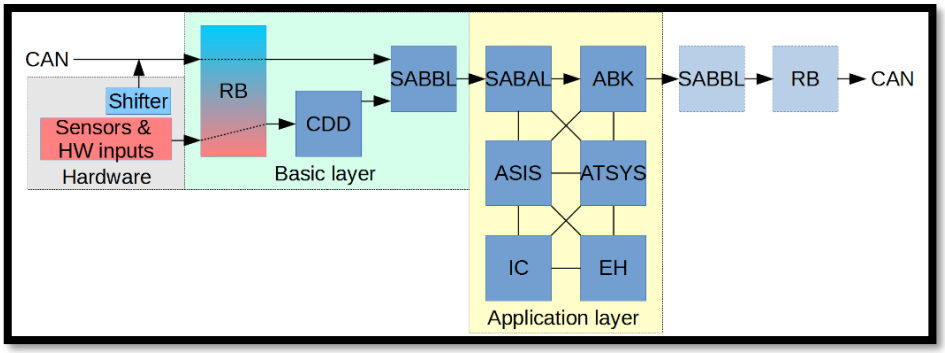
Software je nahrán v řídící jednotce převodovky a zasahuje do značné části celkového řízení automobilu. Řídící jednotka je představena na obrázku 5 Řídící jednotka automatické převodovky. Za tímto účelem je software v řídící jednotce rozdělen do struktury podle jednotlivých modulů, které plní požadovanou funkci.

Obrázek 5 Řídící jednotka automatické převodovky

Zdroj: ZF Engineering Plzeň s.r.o.

Softwarová struktura řídící jednotky s jednotlivými moduly je vyobrazena na obrázku 6 Softwarová struktura řídící jednotky automatické převodovky. Veškerá vnitřní komunikace mezi řídícími jednotkami probíhá po sběrnici. V automobilech se používají sběrnice typu CAN, CAN FD a FlexRay podle výrobce automobilu.

Obrázek 6 Softwarová struktura řídící jednotky automatické převodovky



Zdroj: ZF Engineering Plzeň s.r.o.

Řídící jednotka automatické převodovky dostává veškeré informace od řadící páky, senzorů a dalších hardwarových vstupů po CAN sběrnici do základní vrstvy, která zpracovává jednotlivé signály pro další moduly nacházející se v aplikační vrstvě.
V základní vrstvě se nacházejí moduly s funkcemi:

- RB: zpracování CAN signálů,
- CDD: signály čidel napojených přímo do RB,
- SABBL: zpracování signálů pro další moduly je rozdělen na vstupní a výstupní část.

Aplikační vrstva zajišťuje veškeré výpočty, operace a řízení na základě dat obdržených ze základní vrstvy.
V aplikační vrstvě se nacházejí komponenty s funkcemi:

- ABK: rozhraní pro interakci s řidičem, v současné době se nahrazuje komponentou DTI,
- ASIS: adaptivní řazení (řadicí programy – ECO, SPORT…),
- ATSYS: řízení řadicích prvků,
- SABAL: aplikační vrstva SAB,
- EH: komponenta starající se o chyby (kontrola, průběh chyby a její zpracování),

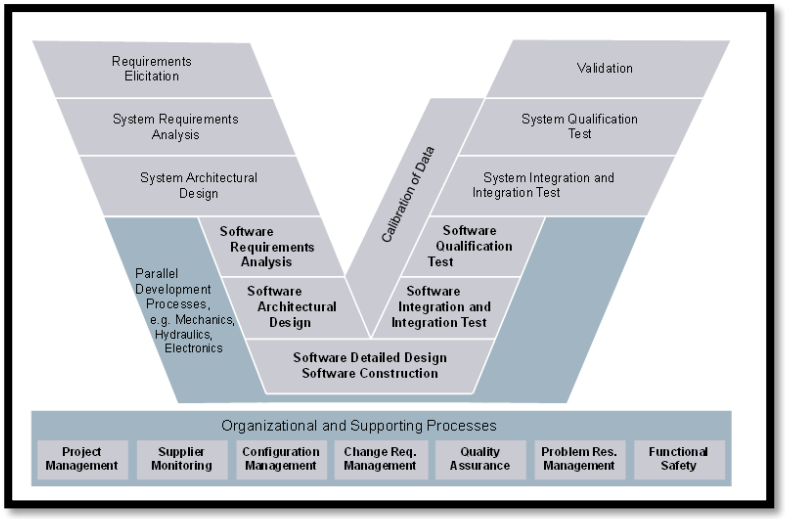
- IC: používá se při vývoji jako mezikrok, pokud není nová funkcionalita plně implementovaná ve všech komponentách.

Po zpracování všech dat v aplikační vrstvě jsou signály zpracovány výstupní komponentou SABBL, RB a dále posílány po CAN sběrnici do dalších řídících jednotek a komponentů automobilu.

4.1.2 Současný proces testování v ZF Engineering Plzeň s.r.o.

ZF Engineering Plzeň s.r.o. po celou dobu životního cyklu vývoje softwaru spolupracuje s týmy nacházející se v zahraničních lokací po celém světě. Pro daný způsob vývoje softwaru je nevhodné použití agilní metodiky, jelikož na softwaru pracuje velké množství pracovníků v různých časových pásmech. Proto je nezbytné zvolit tradiční metodiku vývoje životní~~ho~~ cyklu softwaru. Tradiční metodika je zastoupena W-modelem, jelikož je v něm kladen důraz právě na testování již v rané fázi vývoje softwaru a v průběžné spolupráci testerů při jeho vývoji. W-model je představen na obrázku 7 W-model ZF.

Obrázek 7 W-model ZF



Zdroj: ZF Engineering Plzeň s.r.o.

Tato diplomová práce bude zaměřená na procesy testování v oddělení DITWZ26, které se zabývá aplikačním testováním softwaru řídících jednotek automatických převodovek do automobilů se spalovacími a hybridními motory. Oddělení je rozděleno do dvou týmů BMW a RoW.

Nově vzniklé oddělení DIWTZ 26 přešlo do režimu sdílených pracovních míst, tudíž členové oddělení již nedisponují svým pracovním místem, ale místa jsou sdílená a rezervují se přes rezervační systém. V současné době má každý člen oddělení povinnost se dostavit do kanceláře čtyři dny v měsíci a zbylé dny má možnost pracovat z domova. Režim sdílených míst nese určité úskalí, jelikož je nutné pro správný chod týmů a testování, aby každý den alespoň jeden člen z týmu byl přítomný v kanceláři a v případě potřeby mohl připojovat řídící jednotky a příslušenství na HiLech dle potřeby testování.

Procesy v obou týmech jsou shodné a vyskytují se nepatrné odchylky, které nemají zásadní vliv na testování. Z tohoto důvodu bude dostačující v této diplomové práci provést analýzu pouze jednoho týmu, konkrétně tým RoW. V týmu bude provedena analýza, vyhodnocení současného procesu testování a následný návrh optimalizace procesu testování, který bude moci být aplikován i v celém oddělení DITWZ26.

Tým RoW provádí testování softwaru pro celosvětově známé výrobce automobilů značky Nissan, Renault, Land Rover, Jaguar a méně známé čínské výrobce automobilů. Pro testování řídících jednotek je využívána HiL laboratoř, ve které se nacházejí systémy HiL typů mini HiL, PHS a Scalexio. Systém HiL bude podrobněji představen v kapitole 4.1.3 Nástroje a hardware k testování

Tým RoW se zabývá testováním softwaru pro následující typy převodovek dle projektů:

- 8HP první generace pro Jaguar a Land Rover
- 8P druhá generace pro hybridní pohony Jaguar a Land Rover
- 8HP3G třetí generace pro Jaguar, Land Rover a čínské automobilky
- 8HP4G a 8P4G čtvrtá generace pro Jaguar, Land Rover, čínský FAW
- 8HP50 druhá generace pro čínské automobilky
- 9HP druhá generace pro Jaguar a Land Rover
- 9HP50 druhá generace pro Nissan, Renault a čínské automobilky

Typ řídící jednotky je závislý na generaci konstrukce převodovky a na daném projektu.

Tým RoW se skládá ze sedmi testerů a jednoho koordinátora. Na základě požadavků softwarových test manažerů přiřazuje koordinátor jednotlivým členům týmu testovací případy. Tým RoW se zaměřuje na manuální komponentní testování komponent SABBL a SABAL techniky white box. Dále se pak specializuje na automatické regresní testování komponenty DTI technikou black box. Koordinátor obdrží od softwarového test manažera balíček obsahující testovací případy, který se nazývá Job. Na jehož základě přiděluje testerům jednotlivé testovací případy k otestování konkrétního projektu s aktuální

softwarovou smyčkou. Testovací případy jsou uvedeny v excelovském dokumentu, který nese název Package Planning. V dokumentu jsou sloupce, ve kterých jsou uvedeny testované komponenty a název testovacího případu, také tester, který bude provádět testovaní, a dále je uveden recenzent, který bude provádět revizi, zda vytvořené testy dostatečně pokrývají testovací případ a splňují všechny náležitosti. V dokumentu jsou uvedeny další sloupce s počtem hodin, které jsou k testovacímu případu přiděleny také datumem termínu dokončení, následují čísla Job a RCR nebo CR, které se vážou k testovacím případům, dále stav, v jakém se testovací případ nachází. Stavy jsou klasifikovány v procesu, v revizi, dokončeno, zrušeno a pozastaveno. V dalším sloupci je uveden postup prací na testovacím případu v procentech a v posledním sloupci jsou uvedeny doplňující poznámky k testovanému případu. Package Planning slouží pro přehledné zadávání úkolů testovacích případů a také pro přehled postupů prací na přidělených testovacích případech.

Účtování zákazníkům za testování softwaru je provedeno automaticky prostřednictví nástroje SAP. Ke každému projektu je přiřazen PSP element, který udává, o jaký projekt se jedná a zda bylo provedeno automatické nebo manuální testování. Každý tester zadává PSP element podle typu testování, ke kterému přiřadí počet hodin strávených testováním na daném projektu.

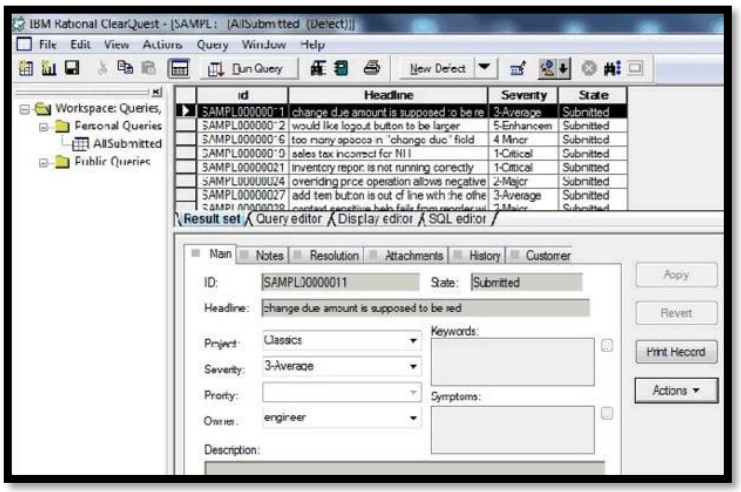
Při testování softwaru se účtuje čas testera, který vynaloží čas strávený testováním testovacího případu a dále se účtuje využívání systému HiL podle délky rezervace v rezervačním systému.

Nástroje k řízení procesu testování:

- ClearQuest: jedná se o nástroj od společnosti IBM, který slouží pro zadávání testovacích případů. Nástroj lze používat v režimu desktopové nebo webové aplikace. V nástroji ClearQuest tester získá k otestování potřebné informace, ve kterých je popsáno, na základě, čeho byl testovací případ vytvořen, je-li to na základě zákaznického požadavku, vzniku nebo úpravy komponenty či na základě nálezu defektu. Jedná-li se o zákaznický požadavek je uveden v testovacím případu odkaz na příslušný modul, kterého se testování týká. Příslušný modul je uveden v nástroji DOORS v sekci tzv. LeMe, kde jsou sepsány zákaznické požadavky k jednotlivým modulům a komponentám ke konkrétním projektům a softwarovým smyčkám. Jedná-li se o vznik a úpravu komponenty, obsahuje ClearQuest přílohu dokumentu, ve kterém je změna zaznamenána. Jedná-li se o testování na základě nálezu defektu, je v ClearQuest uvedeno číslo a odkaz na daný nález v ticketovacím nástroji

Omnitracker. V nástroji ClearQuest je také uvedeno, v jakém stavu se testovací případ nachází. Po provedeném otestování případu se v nástroji ClearQuest případ uzavře s patřičným výsledkem. Dále k testovacímu případu je vložena dokumentace obsahující jednotlivé testy, které jsou opatřeny výsledky. Na obrázku 8 Prostředí ClearQuest je náhled prostředí nástroje ClearQuest, které je využíváno napříč celou společností ZF Group.

Obrázek 8 Prostředí ClearQuest

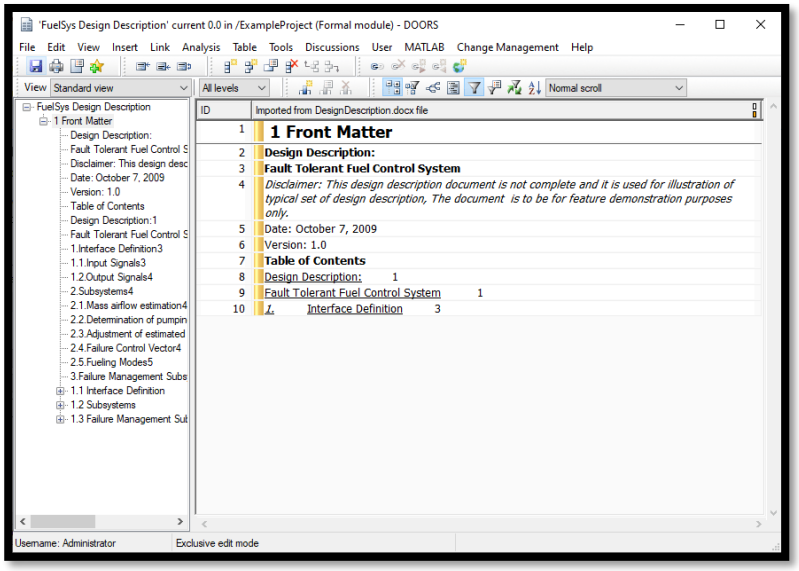


Zdroj: ZF Engineering Plzeň s.r.o.

- DOORS: jedná se o desktopovou aplikaci společnosti IBM. Jak již bylo uvedeno v předchozím textu, tento nástroj slouží ke správě softwarových požadavků, které jsou tvořeny na základě zákaznických požadavků. Softwarové požadavky jsou uvedeny v sekci LeMe dle jednotlivých modulů, které se vážou ke konkrétním projektům. Do tohoto nástroje jsou zaznamenávány testy, které jsou provázány se softwarovými požadavky a jsou odkazovány do sekce LeMe dle jednotlivých modulů. Do nástroje DOORS jsou zaznamenávány testy do jednotlivých kapitol podle komponenty, kterou jsou zastoupeny. V DOORS jsou zaznamenávány veškeré manuální a automatické testy všech komponent, které jsou vytvořeny na základě softwarových požadavcích nebo jsou vytvořeny pouze na základě změn ve specifikaci. Tento nástroj obsahuje celou řadu funkcí, jako je export testů do excelového souboru nebo synchronizaci vytvořených testů do nástroje EXAM pro automatické testování, které bude detailněji představeno v kapitole 4.1.3 Nástroje a

hardware k testování. Nástroj DOORS je v podstatě velkou databází všech testů a zákaznických požadavků. Na obrázku 9 Náhled nástroje DOORS je vyobrazené prostředí nástroje DOORS, které je využíváno napříč celou společností ZF Group.

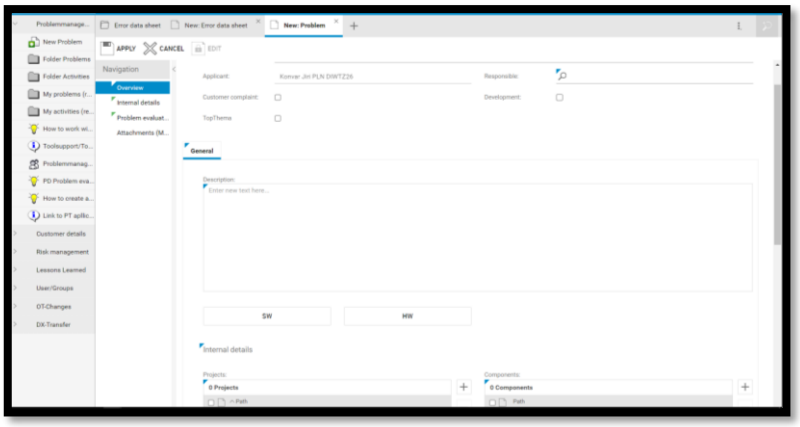
Obrázek 9 Náhled nástroje DOORS



Zdroj: ZF Engineering Plzeň s.r.o.

- Omnitracker: je nástroj určený pro správu defektů odhalených při testování softwaru. Na základě nalezeného defektu je vytvořen ticket, ve kterém jsou důkladně popsány rozdíly mezi očekávaným a skutečným chováním softwaru, které se doloží přiložením naměřených dat, doplňujícími informaci o softwaru a dále je delegován odpovědné osobě. U ticketu lze průběžně sledovat právě v jaké fázi se nachází a způsob řešení defektu. Na obrázku 10 Náhled nástroje Omnitracker je vyobrazené prostředí nástroje Omnitracker, které je stejně jako ostatní nástroje využíváno napříč celou společností ZF Group.

Obrázek 10 Náhled nástroje Omnitracker



Zdroj: ZF Engineering Plzeň s.r.o.

4.1.3 Nástroje a hardware k testování

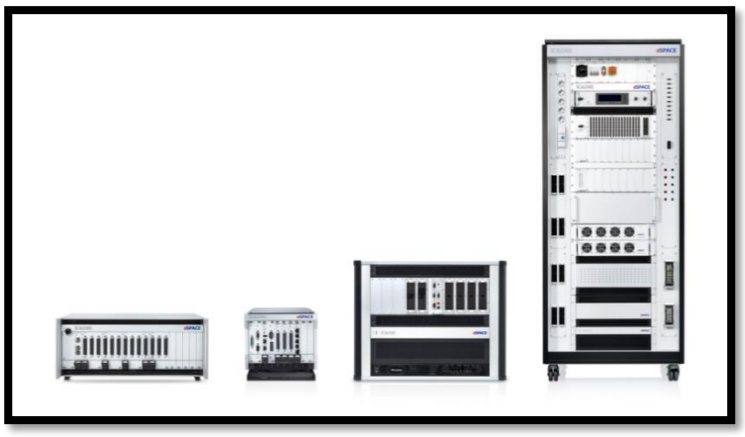
V této kapitole budou představeny nástroje a hardware potřebný pro simulaci potřebného testovacího prostředí. Velká část testování softwaru automatických převodovek se provádí mimo automobil z důvodu snížení finančních nákladů na testování.

Hardware potřebný pro simulaci testovaného prostředí:

- HiL: K simulaci reálného prostředí automobilu v laboratoři se využívá tzv. systém HiL (Hardware in the Loop). Jedná se o sestavu tvořenou z průmyslového PC s operačním systémem Windows. Dále se v sestavě nachází jádro, které obsahuje hydraulické ventily a slouží k běhu konkrétního modelu automobilu. Další zařízení, které se nachází v sestavě je napájecí zdroj, který simuluje palubní síť. Dále sestava obsahuje blok vstupních a výstupních karet, které simulují provozní i poruchové stavy ve vozidle nebo automatické převodovce a dále pak zprostředkovávají komunikaci HiL systému s řídící jednotkou automatické převodovky. Součástí systému je celé příslušenství jako je BoB (Break out box), ke kterému se připojuje řídicí jednotka společně s projektovým konektorem, který simuluje zapojení hydraulických ventilů automatické převodovky konkrétního modelu automobilu. Dalším důležitým příslušenstvím je ETAS box, který zprostředkovává komunikaci se sběrnicemi CAN, CAN FD či FlexRay a dále prostřednictvím ethernetového rozhraní ETK k vyčítaní signálů potřebných při testování softwaru řídící jednotky. Dále se k systému HiL připojuje zařízení Vector VN, kterým se testuje po sběrnici CAN, CAN FD nebo FlaxRay vyčítání servisních údajů o vozidle a řídící jednotce.

Systémy Hil se rozlišují dle konstrukce na mini HiL, PHS a Scalexio. K testování na systému HiL je nutné připojit řídící jednotku s potřebným příslušenstvím a dále již testování probíhá vzdáleně prostřednictvím pracovní stanice testera, který se k HiL systému připojuje pomocí speciálního uživatele. Po veškerém zapojení není již vyžadována přítomnost testera, který si veškeré potřebné stavy nutné pro měření nastaví vzdáleně. Jedná se o zásadní výhodu hlavně pro práci z domova. Před použitím systému HiL je nutné provést rezervaci prostřednictvím rezervačního systému. V rezervačním systému se provádí rezervace HiL systému a řídící jednotky. Dále se při rezervaci vyplňuje čas, kdy se zařízení budou využívat, název projektu, který se bude testovat, dále se zadává název softwarové smyčky, způsob testování a PSP element, který slouží k účtování zákazníkovi za využívání zařízení dle času pro provedení testů. Systém Hil typu Scalexio je vyobrazen na obrázku 11 HiL Scalexio.

Obrázek 11 HiL Scalexio



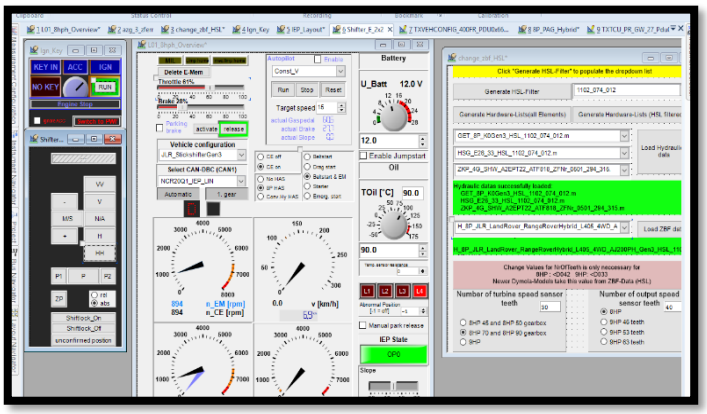
Zdroj: ZF Engineering Plzeň s.r.o.

Nástroje potřebné pro simulaci testovaného prostředí:

- ControlDesk: je aplikace, která simuluje prostředí konkrétního modelu reálného vozu. Dle vybraného modelu je nastaveno jádro systému HiL, jehož nastavení odpovídá testovanému softwaru a řídící jednotce modelu automobilu. V prostředí ControlDesk lze simulovat veškeré jízdní režimy, řazení, jízdu z kopce, do kopce a celou řadu provozních a poruchových stavů jak na automobilu, tak na převodovce. K různým jízdním simulacím se využívají speciální layouty, kterými lze nastavit různé jízdní režimy, provozní a poruchové stavy převodovky. Prostřednictvím

systému HiL lze provádět skutečné zkraty na jednotlivých komponentech převodovky, aniž by došlo k poškození řídící jednotky. ControlDesk obsahuje kompletní databázi veškerých CAN signálů daného modelu automobilu, které lze dle potřeby testování libovolně nastavit, aby byla docílena věrohodná simulace chování skutečného vozu. Prostředí ControlDesk je ukázáno na obrázku 12 Prostředí ControlDesk.

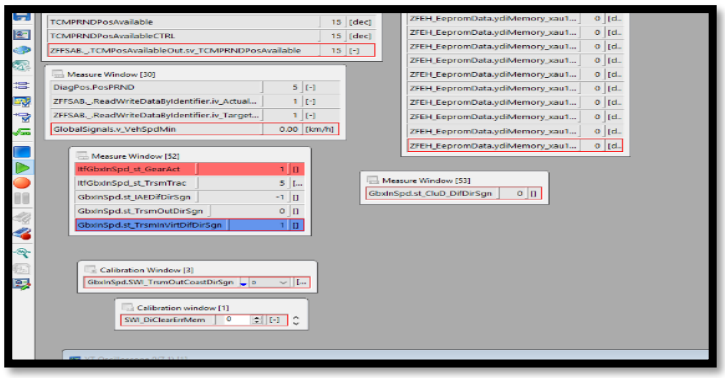
Obrázek 12 Prostředí ControlDesk



Zdroj: ZF Engineering Plzeň s.r.o.

- INCA: jedná se o nástroj určený pro měření dat, který je nutné provozovat současně s aplikací ControlDesk, který zprostředkuje napájení řídící jednotky a zajistí požadované nastavení parametrů testovaného modelu. Měření daného projektu vyžaduje do programu INCA nahrát testovaný software, soubor A2L, obsahující veškeré signály a proměnné nutné pro měření celé softwarové struktury. Dále je nutné vložit CAN databázi CAN signálů pro kontrolu nastavení CAN signálu modelu. Program INCA prostřednictvím ETAS boxu s rozhraním ETK umožňuje v průběžném čase monitorovat všechny potřebné signály, proměnné softwaru a CAN signály, které jsou vybrané v experimentu měření a dále umožnuje vytvořit z měření nahrávku, která pak slouží jako podklad výsledku k testovacímu případu, popřípadě ticketu. Na obrázku 13 Prostředí programu INCA je zobrazeno prostředí s experimentem měření programu INCA.

Obrázek 13 Prostředí programu INCA

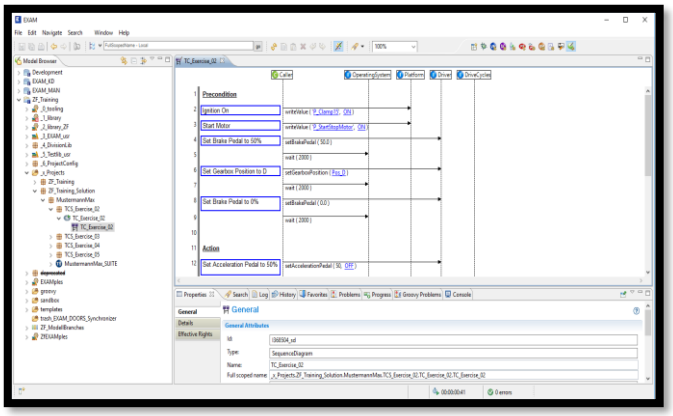


Zdroj: ZF Engineering Plzeň s.r.o.

- EXAM: jedná se o aplikaci, která zajišťuje automatické provedení a vyhodnocení testů. Aplikace byla vyvinuta společností Micronova primárně pro společnost Audi. V současnosti ho využívá celá řada společností zabývající se testováním softwaru v automobilovém průmyslu. Na základě vytvořených testů v nástroji DOORS, které se přenesou po synchronizaci do programu EXAM, je možné provést jejich automatizaci s vyhodnocením v reálném čase za běhu testů nebo až v offline evaluaci po dokončení testů. Program EXAM pracuje společně s programem ControlDesk, který zajišťuje nastavení modelu a s programem INCA, který provádí záznam a nahrávání potřebných dat. Součástí programu EXAM je nutná databáze všech signálů a proměnných testovaných modelů pro vzájemnou spolupráci s nástrojem INCA. Dále pak obsahuje konfigurace jednotlivých testovaných modelů, které jsou potřebné pro vzájemnou spolupráci s nástrojem ControlDesk. V programu se nachází celá řada operací a akcí, které je možné poskládat pro potřeby testování a následné vyhodnocení. EXAM se nejčastěji využívá při regresním testování. Takové testování obsahuje několik stovek testů, které se opakuje v několika softwarových smyčkách testovaného projektu. Také se používá při manuálním testováním, především pro měření testů, které vyžadují velmi rychlé po sobě následující akce v řádech sekund či milisekund. Hlavní výhodou tohoto nástroje je rychlost testování velkého množství testů, a především ušetření času testera při regresním testování, které je možné realizovat v nočních hodinách, jelikož není vyžadována přítomnost testera, protože vše probíhá automaticky. Na obrázku 14 Nástroj automatického testování

EXAM je zobrazen nástroj EXAM, využívající se pro automatické regresní testování.

Obrázek 14 Nástroj automatického testování EXAM



Zdroj: ZF Engineering Plzeň s.r.o.

4.2 Aktuální stav manuálního testování

V této kapitole budou popsány současné procesy spojené s manuálním komponentním testováním projektů čínských značek, Jaguar a Land Rover. Tato kapitola bude rozdělena do podkapitol, ve kterých budou popsány jednotlivé procesy k danému průběhu testování. Tým RoW se specializuje na testování komponent SABAL a SABBL, při které využívá techniku manuálního komponentního testování metodikou white boxu. Tento způsob testování se využívá především k ověření správného chování dané komponenty z důvodu zavedení nové funkce, změny funkčnosti nebo opravy defektu modulu v testované komponentě.

4.2.1 Procesy komponentního testování

Podnět k testování přichází na základě čísla RCR nebo CR. Tester si v nástroji ClearQuest vyhledá dle čísla testovací případ, který nese název obvykle dle testované komponenty nebo modulu. Pod číslem RCR nebo CR jsou uvedeny potřebné informace přiložené ve specifikaci týkající se uvedené změny, kterou je nutné otestovat. Dále zde může být uvedeno číslo ticketu, které testera odkáže do nástroje Omnitracker, ve které je popsán defekt s naměřenými daty odhalený předchozím testováním a je nutné ho přetestovat.

Pro testování projektů Jaguar a Land Rover jsou testerem napsány testy v nástroji DOORS. Následně je provedena revize testů a pak se přechází k samotnému měření dle napsaných testů, které prošly revizí.

Procesně se odlišuje manuální komponentní testování čínských projektů, u kterých se testuje komponenta SABBL technikou rychlých testů na základě specifikace a dokumentace SABBL k danému projektu. Testy se nepíšou do DOORS ani se neprovádí revize testů. Testování se provádí přímo na HiLu, kdy se ověří všechny větve zavedeného nebo upraveného modulu dané komponenty. Dále se kontroluje, zda se proměnné (makra) přenášejí do dalších komponent jako je ASIS, ATSYS, CDD, DTI, SABAL, které jsou určené v dokumentaci Interface Table. Tento způsob testování se u čínských projektů využívá z časových a kapacitních důvodů, jelikož je nutné otestovat velké množství modulů v komponentě SABBL. Tato technika testování je dostatečná, poněvadž po komponentním testování se navazuje na regresní automatické testování, které případně odhalí nedostatky v komponentním testování a již splňuje všechny náležitosti dokumentace.

4.2.2 Příprava specifikace testování

Tento způsob přípravy se používá pouze u projektů Jaguar a Land Rover. Čínské projekty se provádí technikou rychlých testů, které nevyžadují vytvoření testů v nástroji DOORS. Při přípravě testování tester provede uložení specifikace na verzovací disk na intranetu prostřednictvím nástroje ClearCase do složky dle projektu a kapitoly vázané v DOORS. Název složky je shodný s názvem testovacího případu. Složka je opatřena labelem, který označuje smyčku testovaného softwaru. Tento způsob procesu se provádí z důvodu pozdější archivace, aby byly všechny potřebné informace k testovanému případu pohromadě.

Na základě nabitých informací ze specifikace přiložené k testovacímu případu nebo ticketu jsou napsány testy do nástroje DOORS. Pro lepší orientaci a přehled jsou vytvořeny kapitoly jednotlivých komponent dle projektů. V této fázi je přepnut testovací případ v Package Planning do stavu v procesu a dále jsou uvedena procenta postupu.

Testy jsou napsány takovým způsobem, který otestuje všechny stavy, které mohou nastat a otestují se větve v daném modulu komponenty, které jsou spojené se změnou nebo opravou defektu. Pokud to testovací případ vyžaduje, jsou vytvořeny i negativní testy. Po vypracování testů je vytvořena startovací baseline, která zajišťuje verzování změn v modulu pro pozdější zpětnou kontrolu historie. Baseline se označuje identifikačním číslem.

Po vytvoření baseline, tester založí excelovský dokument checklist, ve kterém jsou uvedeny doplňující informace vázané k testovacímu případu. V souboru je uvedeno, zda je přiložená specifikace s odkazem, popřípadě odkaz na softwarové požadavky do modulu LeMe v nástroji DOORS. Dále jsou v souboru uvedeny odkazy k nově vytvořeným testům jednotlivých testovacích modulů a číslo baseline. V této fázi se stav testovacího případu přepíná ze stavu v procesu do stavu revize, který informuje recenzenta o provedení revize.

Po vytvoření checklistu je provedena revize nově napsaných testů, které jsou revidovány určeným recenzentem. Recenzent po provedené kontrole vypracovaných testů vyplní nalezené nedostatky, které jsou rozděleny do třech kategorií dle priority závažnosti na hlavní, vedlejší a ostatní. Hlavní nedostatky jsou klasifikovány jako nedostatečně pokryté testy nebo nepokryté požadavky. Vedlejší nedostatky jsou klasifikovány jako chybně doplněné údaje nebo neuvedené číslo RCR, CR, nevyplněná varianta a další. Ostatní nedostatky jsou klasifikovány především písemnými překlepy. Po opravení všech nedostatků v testech je provedena ukončovací baseline v modulu DOORS, checklist se uzavře a je možné přejít k samotnému testovaní.

4.2.3 Realizace testování

Po přípravě a vytvoření testů je již možné přistoupit k samotnému testování. Nejprve je nutné před samotným testováním zarezervovat HiL a řídící jednotku prostřednictvím rezervačního systému. Rezervuje se HiL Scalexio nebo PHS. Pro projekty Jaguar a Land Rover je nutné rozlišovat, o jakou se jedná generaci, jelikož 3. a 4. generace není na HiL PHS již podporovaná, protože se jedná o již starší zařízení. Další projekty jako Nissan, Renault a čínský FAW 4. generace také nejsou podporované na HiL PHS. Ostatní čínské projekty, Jaguar a Land Rover 1. a 2. generace jsou podporovány pro HiL PHS. Pro měření je nutné na HiL připojit potřebnou jednotku dle projektu, který se bude testovat. Řídící jednotky jsou pro některé projekty společné a rozlišují se připojením projektových konektorů. Po připojení jednotky a veškerého potřebného příslušenství k měření se spustí model v nástroji ControlDesk, kde se vyberou potřebná HSL a ZBF data pro nastavení hydraulických ventilů dle modelu vozu. Dále se zvolí typ řadící páky, vybere se CAN databáze pro daný model vozu a poté se volí hybridní nebo spalovací pohon. V posledním kroku se zapne napájení 12 V a aktivuje se zapalování, kterým se řídící jednotka probudí.

Nyní se spustí program INCA, kterým se do jednotky nahraje testovaný software společně se souborem A2L, který obsahuje databázi signálů a proměnných. Tyto soubory se

nachází na intranetu v projektové složce dané softwarové smyčky společně s celkovou dokumentací. Po nahrání softwaru se otevře měřící experiment, kde si tester ověří, že software správně komunikuje s modelem v nástroji ControlDesk a při jízdě modelu se nevyskytují žádné chyby, které by mohly negativně ovlivňovat měření. Po ověření bezchybnosti se přechází k samotnému měření. Tester si do experimentu vloží signály, proměnné, switche, kalibrace, vstupní a výstupní CAN signály a makra potřebná k měření prostřednictvím souboru A2L, který je přímo dán ke konkrétnímu projektu se softwarem a nahrává se přímo v nástroji INCA. Nyní se již provádí samotné měření na základě testů napsaných v nástroji DOORS nebo technikou rychlých testů dle jednotlivých větví vývojového diagramu a změn ve specifikaci. Veškeré měření je nutné nahrávat z důvodu pozdějšího vyhodnocení. Některé naměřené stavy či přechody jsou méně zřetelné, proto je nutné jejich vyhodnocení ověřit prostřednictvím nástroje MDA, který obsahuje osciloskop. Další důvod vytvoření nahrávky je v případě doložení výsledků zákazníkovi, dále také k archivaci. Nahrávky měření jsou složeny z jednotlivých sekvencí, které jsou rozděleny podle vytvořených testů nebo větví vývojových diagramů.

4.2.4 Zpracování výsledku testování

Po provedení měření tester vyhodnocuje jednotlivé sekvence v průběhu měření nebo pak později prochází jednotlivé nahrávky a provádí vyhodnocení. U manuálního testování záleží na testerovi, aby správně pochopil výsledky jednotlivých testů a správně vyhodnotil celkový výsledek testování. Všechny nahrávky z měření se ukládají do projektové složky testované softwarové smyčky na síti pod názvem testovaného případu.

Po vyhodnocení výsledků se provede u manuálních testů export z nástroje DOORS jednotlivých testů do uceleného excelového souboru. V excelovém souboru se u každého testu vyplní výsledek „prošel nebo selhal“. Pokud se vyskytne i jeden test, který má výsledek „selhal“ je celý testovací případ označený výsledkem „selhal“. Excelovský soubor se dále ukládá na verzovací disk do projektové složky testované softwarové smyčky pro pozdější archivaci. Dále je excelovský soubor odeslán emailem do nástroje NeKeDa, který zpracovává pokrytí projektů softwarové smyčky jednotlivými testy na základě vytvořených softwarových požadavků. Tato část procesu zpracování exportu excelovského souboru se neprovádí u rychlých testů pro čínské projekty.

V případě výsledku testovacího případu „selhal“ je vytvořen ticket, popřípadě tickety v nástroji Omnitracker. Ticket obsahuje informace, kterého projektu a softwarové smyčky

nalezený defekt se týkal a dále jsou zde popsány rozdíly mezi skutečným a očekávaným chováním a jako podklad jsou přiložena naměřená data. V ticketu je uveden vlastník, tedy tester, který ticket zakládal; a ticket deleguje diagnostickému koordinátorovi podle komunikační matice, který je zodpovědný pro daný projekt. Každý ticket má svoje identifikační číslo, které se zadává do excelovské tabulky s dalšími informacemi pro lepší přehled.

Výsledek testování „prošel“ nebo „selhal“ se zadává společně se jménem testera, aktuální softwarovou smyčkou, s počtem hodin potřebných k testování, přiloženým excelovským souborem s výsledky jednotlivých testů, a v případě ticketu jeho číslo do nástroje ClearQuest pod číslem, který se váže k testovacímu případu. Po vyplnění všech údajů se testovací případ uzavře pro daný projekt.

Posledním krokem je vyplnění souboru Test Loop Progress, ve kterém jsou uvedené jednotlivé testovací případy vázané k testovacímu balíku Job pro danou softwarovou smyčku. K jednotlivým testovacím případům se vyplňuje, zda se nachází na verzovacím disku ClearCase specifikace se změnami opatřená labelem a dále jestli byla provedena revize testů s udaným počtem nalezených hlavních a vedlejších chyb při revizi, dále se vyplňuje, zda byl vytvořen export z DOORS, provedení baseline, jaký je výsledek testovacího případu a zda je uzavřen testovací případ v nástroji ClearQuest. Tento excelovský soubor se také nachází na verzovacím disku v nástroji ClearCase v projektové složce se softwarovou smyčkou. V posledním kroku se provede přepnutí testovacího případu do stavu dokončeno a zadání postupu na 100 %.

Po otestování celého projektu na dané softwarové smyčce se veškerá dokumentace, software, naměřená data, specifikace, exporty testů a Test Loop Progress archivuje pro případnou pozdější kontrolu. Po provedené archivaci jsou testované případy odstraněny ze souboru Package Planning.

4.3 Aktuální stav automatického regresního testování

V této kapitole budou popsány současné procesy automatického regresního testování. Představeny budou pouze procesy, které se odlišují od manuálního komponentního testování, jelikož velká část procesů by se opakovala.

4.3.1 Procesy automatického testování

Tým RoW se zabývá automatickým regresním testováním modulů komponenty DTI u čínských projektů, které používají automatické převodovky 8HP50, 8HP3G a 9HP50.

DTI komponenta je rozdělena do základních a rozšiřujících modulů, které zajišťují celou řadu operací a procesů spojených s interakcí řidiče s převodovkou. Základní modul „POŽADAVEK ŘIDIČE“ komponenty DTI zajišťuje komunikaci řadící páky a zadávání pozic automatické převodovky, které jsou P, R, N, D a M. Následující komponenta „JÍZDNÍ REŽIM“ slouží k nastavení základních jízdních režimů automatické převodovky, které jsou normální, manuální a sportovní režim. Dle v modelu automobilu mohou být základní jízdní režimy doplněny o rozšířené jízdní režimy, do kterých patří eko a komfortní režim, písek, kamení, sníh. Jízdní režimy slouží k nastavení volby mezi režimy pohonu 2x4 nebo 4x4, pokud takovým pohonem model automobilu disponuje. Dále komponenta DTI obsahuje modul „DISPLEJ“, který zprostředkovává řidiči informace o zařazené pozici a jízdním režimu prostřednictvím palubní desky. Dalším důležitým modulem je „UPOZORNĚNÍ ŘIDIČE“, které také prostřednictvím palubní desky informuje řidiče o chybných a poruchových stavech nejen automatické převodovky, ale i ostatních stavech automobilu. Posledním základním modulem DTI komponenty je „DETEKTOR PŘÍTOMNOSTI ŘIDIČE“, který ověřuje stavy spojené s přítomností řidiče ve vozidle. Základní moduly se nacházejí ve všech modelech projektů a liší se pouze jejich funkcemi. Rozšiřující moduly komponenty DTI jsou obsaženy pouze u vybraných modelů projektů, které vyžadují speciální funkce. Do kategorie rozšířeného modulu patří „VZDÁLENÝ START“, který umožňuje nastartovat vozidlo na dálku, aniž by byl řidič přítomen ve vozidle. Další rozšiřující modul je „DAS“ (SYSTÉM ASISTENTA ŘIDIČE), který zajišťuje funkce spojené s automatickými asistenčními režimy pro řidiče. Rozšiřujících modulů je celá řada, záleží na přáních a požadavcích zákazníka.

4.3.2 Příprava specifikace testování

Při automatickém regresním testování obdrží koordinátor týmu Job k otestování celé DTI regrese, kterou rozdělí podle komponent mezi testery v týmu. Tester obdrží testovací případ k otestovaní pod názvem daného modulu. Modul je pak následně vyhledán v nástroji DOORS v kategorii LeMe pod kapitolou nesoucí shodný název s modulem. V LeMe jsou uvedeny softwarové požadavky vytvořeny specifikátory, kteří vycházejí ze zákaznických požadavků. Softwarové požadavky popisují, jak se má daná komponenta či modul chovat,

jakou má mít funkci a pro jakou variantu je platná. Někdy bývá Job doplněn také o čísla RCR a CR, která upřesňují chování, zavádí novou funkci nebo provádí opravu daného modulu. Na základě softwarových požadavků, RCR a CR jsou vytvořeny testy, které jsou propojeny s příslušným modulem. Na vytvořené testy je provedena revize stejně jako je tomu u manuálních testů. Po revizi již následuje proces samotného testování.

4.3.3 Realizace automatického testování

Proces samotného tetování začíná v automatizaci vytvořených testů v nástroji EXAM. Testy vytvořené v nástroji DOORS jsou nejprve synchronizovány do nástroje EXAM v obdobné struktuře, která se nachází v nástroji DOORS. Po provedené synchronizaci je možné přistoupit k samotné automatizaci. Automatické testy jsou vytvořeny ze sekvencí, které obsahují potřebné operace. Automatizace má danou strukturu sekvencí, kterou je nutné dodržovat a je rozdělena do tří částí, jako jsou přípravná sekvence, dále pak sekvence akcí a sekvence vyhodnocení. V úvodní části je vytvořena přípravná sekvence, která obsahuje celkem tři sekvence. První je měřící sekvence, ve které jsou uvedeny veškeré signály a proměnné, které je nutné sledovat a zaznamenávat pro potřeby měření. Po ní následuje inicializační sekvence, která slouží k nastavení inicializace a následnému ovládání nástrojů ControlDesk a INCA, podle následujících sekvencí akcí. Třetí sekvence nastavuje výchozí počáteční stav modelu vozu před sekvencí akcí. Nyní se nastavuje sekvence akcí, která může obsahovat jakékoliv potřebné množství akcí, operací a nastavení podle jednotlivých modelů. Po provedené sekvenci akcí je vytvořena třetí sekvence vyhodnocení, která obsahuje sekvenci předpřípravu vyhodnocení, která uloží veškeré naměřené hodnoty a je povinná. Po ní již přichází offline vyhodnocení měření na základě stanovených podmínek. Offline sekvencí může být opět jakýkoliv počet podle potřeb měření. Závěrečná sekvence je tzv. uklízecí sekvence, která vrátí model do základního nastavení pro následující měření a je také povinná.

Po automatizaci testů je již možné pustit zautomatizované testy. U automatického testování je nutné stejně jako u manuálního testování mít na systému HiL spuštěný a nastavený model prostřednictvím nástroje ControlDesk a zapnuté měření v nástroji INCA. Program EXAM po provedení inicializace těchto nástrojů již přebírá jejich řízení a ovládání na základě definovaných a z automatizovaných testů. Po provedeném otestování jsou testy již opatřeny stavem „prošel“ nebo „selhal“ na základě definovaných podmínek během měření nebo podmínek v offline hodnocení. Testerovi je umožněno si výsledky měření

prohlédnout a následně manuálně vyhodnotit v reportovací databázi, kde jsou nahrány výsledky měření provedených jednotlivých testů. Na základě provedené analýzy výsledků, automatizované testy upraví nebo pokud u konkrétního testu není automatizace možná z důvodu absence operace, funkce nebo signálu, doměří test manuálně a výsledné manuální měření se přiloží k automatizovanému testu a změní se stav výsledku dle manuálně naměřených dat. Takto upravený automatizovaný test je nutné doplnit komentářem, proč tak bylo provedeno.

4.3.4 Zpracování výsledků automatického testování

Zpracování a odevzdání výsledků u automatického testování je shodné jako u manuálního testování. Procesy odlišující se od manuálního testování budou uvedeny v této kapitole.

Po otestování jsou výsledky všech testerů zkopírovány do sandboxu v plzeňské reportovací databázi, ve kterém je vytvořena kolekce obsahující všechny výsledky testů testovaných modulů komponenty DTI. Takto vytvořená kolekce je opatřena názvem dle testované smyčky projektu a je nakopírována do oficiální databáze jako podklady k naměřeným testů a také k archivaci, jelikož jsou z plzeňské databáze pravidelně mazána naměřená data starší 180 dnů, z důvodů zajištění optimální rychlosti databáze. Z vytvořené kolekce jsou exportována data a dokumenty, které jsou dále přiloženy do testovacího balíku Job. Exportují se naměřená data, pdf soubor obsahující všechny podrobné údaje o testech a následně soubor xml. Při automatickém testování se neprovádí export excelovského soboru z nástroje DOORS, jelikož jsou potřebné informace obsaženy v kolekci. Výsledky se odevzdávají do nástroje ClearQuest přímo do testovacího balíku Job, a ne podle názvu testovacích případů jako je tomu u manuálního měření. Na ClearQuest k Job je přiložen odkaz do oficiální databáze na testovanou kolekci, dále je přiložen pdf soubor. Pokud jsou nalezeny defekty, jsou uvedeny v balíku Job. Poslední proces, který se odlišuje od manuálního testování je odeslání emailem soboru xml do nástroje NeKeDa ke zpracování výsledků.

4.4 Analýza současného procesu manuálního testování

V této kapitole bude provedena analýza na základě popsaných procesů v kapitole 4.2 Aktuální stav manuálního testování a metodikou pozorování spojené s testováním v oddělení

DITWZ 26 týmu RoW. Analýza procesu manuálního komponentního testování, na jejichž základě bude zpracován návrh optimalizace procesů manuálního testování.

4.4.1 Analýza procesů komponentního testování

Analýzy procesů komponentního testování zahrnují procesy spojené s přidělením testovacích případů jednotlivým testerům. Koordinátor týmu testerů obdrží Job k otestování od softwarového test manažera prostřednictvím nástroje ClearQuest. Job koordinátor rozdělí na jednotlivé testovací případy a přidělí testerům k otestování prostřednictvím excelovského souboru Package Planning.

V této části procesu vzniká problém na straně softwarového test manažera, který zadává Job k otestování s nedostatečnou časovou rezervou. Dále je nutné zahrnout také čas potřebný pro rozdání jednotlivých testovacích případů koordinátorem, které způsobí ještě delší časovou prodlevu, než je možné začít testovat. Testeři tudíž mají nedostatečný časový prostor k otestování jednotlivých testovacích případů. Na základě krátkých testovacích termínů jsou členové testovacího týmu zbytečně vystaveny nežádoucímu tlaku, který může způsobit nedostatečné otestování případů. Nedostatečné otestování může vést k neodhalení defektů, které mohou způsobit nežádoucí chování softwaru a pozdější odhalení defektů je spojeno s vyššími náklady na jejich odstranění. Nejčastěji dochází k pozdnímu dodání testovacích balíčků Job zejména u čínských projektů v porovnání s projekt Jaguar a Land Rover, jelikož čínské projekty obsahují značné množství variant a rozsáhlé množství komponent k otestování a také se jedná o větší podíl testovacích prací v poměru pěti čínských projektů ku dvěma projektům Land Rover a Jaguar.

Faktory ovlivňující pozdní dodání testovacích balíku Job:

1. Hlavním faktorem ovlivňující pozdní dodání testovacího balíku Job k otestování je zpožděn ze strany vývoje softwaru, kdy vývojové oddělení nedodrží harmonogram a software je k otestování pozdě uvolněn, což je způsobeno zkrácením časového úseku mezi vydáním jednotlivých softwarových smyček. Pozdní uvolnění softwaru je způsobeno nedostatečnou kapacitou vývojového týmu nebo opravou velkého množství defektů nalezených v předchozí softwarové smyčce.

2. Dalším faktorem ovlivňující časové prodlení jsou pozdě obdrženy zákaznické požadavky, ze kterých vycházejí testovací požadavky potřebné pro testování.

3. Vedlejším faktorem přispívajícím k pozdnímu dodání testovacího balíku Job je způsobeno, že se softwarový test manažeři nacházejí na odlišných geografických místech, jako je tomu u čínských projektů, kde je v časovém posunu rozdíl o 6 až 7 hodin SEČ. Časový posun také negativně ovlivňuje komunikaci.

4. Dalším méně častým faktorem je přímo pochybení ze strany softwarového test manažera, který opomene testovací balík nebo jednotlivé testovací případy dále delegovat.

4.4.2 Analýza přípravy specifikace testování

Analýza přípravy specifikace testování bude rozdělena na dvě analýzy. První analýza se bude zaměřovat na čínské projekty a druhá analýza se bude zabývat projekty Jaguar a Land Rover.

Analýza přípravy specifikace čínských projektů:

V procesu techniky rychlých testů není zásadní nedostatek. Příprava probíhá velice rychle, bez velké administrativní zátěže testerů. Všechny potřebné specifikace a dokumentace jsou vždy dostupné v nástroji ClearQuest nebo v projektové složce.

Faktory ovlivňující proces testování:

1. Faktorem je zadání testovacího případu, který tým RoW není schopen otestovat z důvodu odbornosti speciální komponenty nebo absence potřebného nástroje k otestování.

2. Faktor, který ovlivňuje procesy testování jsou testovací případy, které nejsou relevantní pro testovanou variantu. Obvykle se vyskytne případ s nerelevantní variantou při zavádění nové komponenty, která není dle dokumentace k softwaru v testované variantě zavedená. Také se zřídka vyskytuje testovací případ, který se zabývá otestováním databáze, ve které se dle dokumentace databáze CAN signály nevyskytují.

Výše uvedené body sice spadají do náplně v oblasti testování a neřadí se přímo k zásadním faktorům, které by komplikovaly nebo ztěžovaly testování, ale omezením těchto faktorů dojde ke zvýšení efektivity testování, které bude mít za následek snížení nákladů.

Výhody techniky rychlých testů:

- Technika rychlých testů přináší výhody rychlého testování bez zatěžování testerů s dlouhavou administrativou.
- Snížení časové náročnosti na provedení testovacích případů.
- Snížení nákladů na testování.

Nevýhody techniky rychlých testů:

- Nevýhoda spočívá v absenci písemné dokumentace testů a následné revizi vytvořených testů. V tomto případě je kladen vyšší důraz na zodpovědnost testerů, že testovací případ správně pochopili a komponentu vhodně otestovali.
- Lze využít pouze u testování méně složitých komponent SABBL.

Analýza přípravy specifikace projektů Jaguar a Land Rover:

Proces přípravy je u projektů Jaguar a Land Rover náročnější z hlediska administrativy, jelikož se testovací případy zabývají nejenom testováním komponent SABBL, ale SABAL, DTI, která vyžadují vytvoření dokumentace testů spojené s následnou revizí vytvořených testů a archivaci specifikace, která je součástí testovacího případu.

Faktory ovlivňující proces testování:

1. Významný faktor, který zásadně ovlivňuje proces přípravy specifikace je, že testovací případ nelze otestovat, jelikož se jedná o nerelevantní testovací případ pro testovanou variantu. Problém s neplatnou variantou se vyskytuje přibližně ve 14 %. Faktor je způsobený především společným základním jádrem softwaru, ze kterého vycházejí všechny varianty modelů a podle daného modelu se odvíjejí další komponenty a funkčnosti. Zásadní rozdíly jsou mezi variantami první, třetí a čtvrtou generací převodovek a mezi variantami se spalovacími motory a s hybridními pohony, kde se používá odlišný typ řídících jednotek. První až třetí generace používá komponentu ABK, ale čtvrtá generace již využívá komponentu DTI. Komponenty DTI a ABK se od sebe odlišují funkčností a rozdíl je i v komunikaci mezi navazujícími komponentami. Nastává situace, kdy nalezený defekt modulu určité komponenty, který byl již na dané variantě otestován. Je požadováno tento nalezený defekt otestovat i na zbylých variantách, i když není ověřeno, že testovaná varianta obsahuje funkci nebo modul v dané komponentě. Hlavní problém spočívá v tom, že ne všechny případy softwarový test manažer dokáže analyzovat, zda je testovací případ relevantní. V tomto případě tester tráví značnou část času hledáním

v dokumentaci a komunikací, jak funkčnost nebo modul komponenty otestovat. Ve většině případů dojde ke zjištění, že modul či funkčnost testovaný modul neobsahuje. Časová náročnost identifikace testovatelnosti se velmi často rovná času vyhrazenému k celému otestování. Základ problému je na straně specifikátorů, kteří nedostatečně delegují potřebné informace o testovacích případech softwarovým test manažerům a dále je způsoben nedostatky již v zákaznických požadavcích.

2. Méně významný faktor, který ovlivňuje proces přípravy specifikace je shodný jako u čínského projektu, kdy dochází k tomu, že tým RoW není schopný otestovat požadovanou funkčnost či modul komponenty z důvodu absence potřebného nástroje či příslušenství nezbytného k otestování. V tomto případě se jedná především o testování servisních protokolů. Dalším důvodem, který neumožňuje otestování je velmi specifická funkčnost, která vyžaduje speciální znalosti testované komponenty jako jsou řadící strategie nebo cyber security.

4.4.3 Analýza realizace manuálního komponentního testování

Analýza procesu samotné realizace testování se zabývá vyhledáváním faktorů, které negativně ovlivňují procesy spojené s přípravou měření a již samotným měřením. Procesy jsou shodné jak u čínských projektů, tak u projektů Jaguar a Land Rover, proto budou analyzovány společně.

Faktory ovlivňující realizaci testování:

1. Faktorem je rezervace HiLu, která se realizuje rezervačním systémem. HiLy jsou celkově dost vytíženy z důvodu využívání více oddělení a jejich rezervace je nutné provádět s větším časovým předstihem. Nejvíce jsou exponovány HiLy Scalexio, jelikož jsou novější a umožňují měřit všechny projekty a varianty. Tým RoW Scalexio využívá pro měření projektů Jaguar a Land Rover automatických převodovek 3. a 4. generace a čínské varianty FAW automatické převodovky 4. generace. Na zbylé čínské projekty, Jaguar a Land Rover automatických převodovek 1. a 2. generace je možné využít starší HiLy PHS. V současnosti jsou dlouhodobě využívány HiLy PHS v plzeňské pobočce čínským testovacím oddělením. To má za následek celkovou vyšší vytíženost systému HiL. Z celkového počtu dvanácti PHS HiLů je obsazeno šest PHS HiLů čínským oddělením po celý den od 1:00 hod do 18:00 hod SEČ, i přestože HiLy nejsou po celou dobu využívány. Poskytnutí HiLů

pro čínské oddělení je snížena kapacita testování pro plzeňské oddělení u PHS HiLů až o 50 %.

2. Faktorem je využití řídících jednotek. Vychází z 1. faktoru poskytnutí k testování PHS HiLů čínským oddělením, které využívají plzeňské řídící jednotky pro čínské projekty. Z důvodu obsazení až 50 % řídících jednotek je snížena testovací kapacita plzeňského oddělení. Dochází k situaci kdy, tým RoW nemá volnou řídící jednotku 8HP2G, 8HP3G nebo 9HP50. Řídící jednotky se rezervují prostřednictvím HiL rezervačního systému společně s HiLem. Často se stává, že testeři z čínského oddělení si společně s HiLem nerezervují řídící jednotku, a to má za následek neodpovídající počet zarezervovaných jednotek se skutečně používanými jednotkami.
   Plzeňská laboratoř disponuje řídícími jednotkami pro čínské projekty:
   - 8HP50: 5 kusů
   - 8HP3G: 5 kusů
   - 9HP50: 3 kusy

3. Faktor se týká nefunkčního modelu nástroje ControlDesk k testování softwaru. Tento problém se týká softwaru jak u čínských projektů, tak u projektů Jaguar a Land Rover. Součástí testovacího balíku Job je testovací software, ke kterému by mělo být vydáno nastavení modelu dle aktuální softwarové smyčky. Nastavení modelu většinou není aktuální a při zkušební jízdě modelu je zjištěno, že model vozu se chová neobvykle nebo jízdu neumožňuje vůbec. Tester by měl svůj čas věnovat primárně testování, ale místo toho věnuje neadekvátně dlouhý čas zprovoznění modelu na testované softwarové smyčce. Nefunkční model je ve většině případů zapříčiněn chybným nastavením z důvodu absence aktuálního nastavení modelu pro danou softwarovou smyčku. V krajních případech se jedná o dodání přímo vadného softwaru, který obsahuje chybné kalibrace, a tudíž je pro testování nepoužitelný.

Zbylé procesy spojené se samotným testováním již nejsou narušené faktory, které by měly zásadní vliv na omezení testování.

4.4.4 Analýza zpracování výsledku manuálního komponentního testování

V této podkapitole bude provedena analýza faktorů, které budou vycházet z procesů popsaných v kapitole 4.2.4 Zpracování výsledků testování, které negativně ovlivňují procesy se zpracováním výsledků testování. Procesy zpracování výsledků jsou u projektů Jaguar, Land Rover a čínských variant podobné. U čínských projektů, které se provádí technikou rychlých testů odpadá export excelovského souboru testů z nástroje DOORS a s ním spojené úkony. Z tohoto důvodu jsou procesy se zpracováním výsledků u projektu Jaguar a Land Rover spojeny s větším množstvím administrace oproti čínským projektům.

Faktory negativně ovlivňující proces zpracování výsledků testování:

1. Zásadním faktorem při realizaci procesu zpracování výsledků testování je absence ucelenosti v jednom nástroji. V současné době je zvlášť nástroj Omnitracker na správu ticketů, oddělený nástroj ClearQuest na správu testovacích případů, zvlášť nástroj NeKeDa na zpracování výsledků testů a separátně program DOORS na správu a dokumentaci testů a softwarových požadavků. Po administrativní stránce se jedná o zdlouhavý proces. Tester po změření musí vytvořit export testů z nástroje DOORS a jednotlivé testy v souboru označit výsledkem. Export excelovského souboru se nahrává do nástroje ClearQuest k patřičnému testovacímu případu a znovu se zadává výsledek a dále se pak soubor ukládá na verzovací disk do projektové složky testované softwarové smyčky. Dále se naměřená data ukládají na pevný disk na intranetu do projektové složky testované softwarové smyčky. V případě odhalení defektu je nutné vytvořit ticket v nástroji Omnitracker, ke kterému se opět přikládají data. Excelovský soubor se dále posílá emailem do nástroje NekeDa ke zpracování výsledků. V závěru je nutné vyplnit soubor TestLoopProgress, ve kterém jsou uvedeny základní informace o provedení testování testovacího případu, a to z důvodu velkého množství kroků. Jedná se o proces spojený s velkým množstvím úkonů, který se stává zdlouhavým, neefektivním a nepřehledným.

2. Zásadním faktorem je rychlost a spolehlivost nástrojů ClearQuest a DOORS. Jedná se již o starší nástroje, ke kterým neexistuje technická podpora.
   - Nástroj ClearQuest je možné používat v internetové aplikaci, která je pro uživatele přehlednější, ale vykazuje problémy při ukládání výsledků, kdy se

provedené změny neuloží nebo aplikace celkově spadne z důvodu přetížení serveru. Druhá možnost je použít ClearQuest v režimu desktopové aplikace, která je sice spolehlivější, nevykazuje problémy s ukládáním, ale není přehledná, jelikož nemá optimalizované rozlišení na formát FullHD.

- Nástroj DOORS je desktopová aplikace, která je sice spolehlivá, nevykazuje problémy s ukládáním nebo s nestabilitou systému. Jejími zásadními nedostatky je především rychlost, uživatelská přehlednost a ovladatelnost.

Zbylé postupy spojené s procesem se zpracováním výsledků testováním již nejsou narušené faktory, které by měly zásadní vliv na omezení testování.

4.5 Analýza současného procesu automatického regresního testování

V této kapitole bude na základě popsaných současných procesů v kapitole 4.3 Aktuální stav automatického regresního testování metodikou pozorování a přímo testování v plzeňském oddělení provedena analýza procesu regresního automatického testování softwaru převodovek typů 8HP50, 8HP3G a 9HP50 čínských projektů, na jejichž základě bude zpracován návrh optimalizace procesů automatického regresního testování.

4.5.1 Analýza procesů automatického testování

V případě automatického regresního testování jsou testovací případy rozděleny do jednotlivých modulů, které zajišťují pokrytí testováním celé komponenty DTI.
Faktory ovlivňující procesy:

1. Zásadním faktorem, který negativně ovlivňuje procesy testování jsou nerelevantní a netestovatelné funkčnosti, které jsou zadány prostřednictvím RCR nebo CR, které doplňují softwarové požadavky uvedené v LeMe pro danou komponentu testované varianty. Přibližně u 11 % zadaných RCR nebo CR je požadováno otestovat funkčnost, kterou model varianty neobsahuje. Tím dochází k nežádoucímu časovému využití testera, jelikož se snaží otestovat funkčnost, která není v softwaru obsažena. Tento faktor je způsoben nedostatečným analyzováním testovacího případu dané varianty ze strany software test manažera nebo specifikátorem a přebíráním testovacích případů mezi jednotlivými variantami.

2. Méně významný faktor obsahuje nerelevantní softwarové požadavky uvedené v LeMe pro danou testovatelnou variantu modelu. Tento faktor se objevuje v 5 % testovaných případech a je způsoben obdobně jak je uvedeno u 1. faktoru.

3. Na základě analýzy nebyly dále odhaleny další zásadní faktory, které by významně ovlivňovaly procesy automatického regresního testování.

4.5.2 Analýza přípravy automatického regresního testování

Tato analýza se zabývá selektováním faktorů ovlivňující přípravu a zadávání konkrétních testovacích případů jednotlivým testerům.

Faktory ovlivňující procesy:

Faktor, který narušuje procesy přípravy automatického regresního testování je pozdní přidělení testovacího případu jednotlivým testerům a tím zkrácení času potřebného pro testování. Tento faktor vystavuje testera nežádoucímu tlaku v průběhu testování a může být důsledkem nedostatečného otestování nebo pochybení ze strany testera. Vznik faktoru je způsoben pozdním dodáním testovacího případu koordinátorovi týmu ze strany vývojového týmu prostřednictvím softwarového test manažera.

Zbylé faktory negativně ovlivňující procesy přípravy automatického regresního testování jsou shodné s faktory, které byly uvedeny v předchozí kapitole 4.5.1 Analýza procesů automatického testování.

4.5.3 Analýza realizace automatického regresního testování

Na základě popsaných současných procesů byly zjištěny podobné problémy, které negativně ovlivňují realizaci automatického regresního testování.

Faktory ovlivňující procesy:

Hlavním faktorem ovlivňující realizaci testování, jsou problémy vznikající během automatizace testů v programu EXAM. Jedná se o chybějící operace, absence signálů a proměnných v databázích. Dále pak nesprávné offline vyhodnocení programu a také se zřídka vyskytují časově náročnější implementace a ladění automatizace, než by vyžadovalo samotné manuální testování.

4.5.4 Analýza zpracování výsledků automatického testování

Na základě popsaných současných procesů byly zjištěny obdobné problémy, které negativně ovlivňují zpracování výsledků automatického testování. Faktory byly zanalyzovány a uvedeny v kapitole 4.4.3 Analýza zpracování manuálního komponentního testování.

Faktory ovlivňující procesy:

1. Zásadní faktor, který významně ovlivňuje procesy testování je nesprávně nakalibrovaný software, který způsobuje neočekávané chování softwaru a vznik negativně-pozitivních výsledků. Software obsahuje velké množství různých kalibrací, do kterých jsou zahrnuty různé meze určující rychlost, proudové a napěťové hodnoty v daných funkcích. Chyby v kalibracích vznikají přenosem a kopírováním softwaru mezi jednotlivými variantami modelů projektů. Každá varianta modelu vyžaduje vlastní kalibrace, které jsou uvedeny v zákaznických požadavcích. Nesprávné kalibrace v softwaru generují velké množství defektů, které je nutno reportovat. Vzniklé defekty vyžadují velké množství času na reportování u testerů a na analyzování ze strany diagnostických manažerů. Tento faktor způsobuje vysokou časovou náročnost procesu automatického regresního testování.

2. Faktor je způsoben nesprávnými softwarovými požadavky, které se rozcházejí se zákaznickými požadavky. Dochází k testování funkčnosti, kterou zákazník nepožaduje nebo vyžaduje odlišné chování softwaru, a to způsobuje také vznik negativně-pozitivních testů, které mají za následek také reportování velkého množství defektů. Také tento faktor zapříčiní vysokou časovou náročnost procesu automatického regresního testování.

3. Méně významný faktor se týká značným přetížením oficiální databáze pro ukládání výsledků v programu EXAM, který způsobuje dlouhý export kolekce a výpadky během exportu z plzeňské databáze do oficiální databáze.

4.6 Návrh opatření řešení procesu manuálního testování

V této kapitole budou navrženy optimalizace procesů manuálního komponentního testování na základě provedené analýzy uvedené v kapitole 4.4 Analýza procesu manuálního testování v oddělení DITWZ 26 týmu RoW.

4.6.1 Navrhované řešení procesů komponentního testování

Návrh zlepšení dle jednotlivých faktorů:

1. Návrh řešení pro tento faktor lze těžko ovlivnit ze strany plzeňského oddělení, jelikož se jedná o procesy, které zasahují až do oblasti spojené s vývojem a opravou defektů v softwaru a prochází napříč jinými týmy v odlišných geografických odděleních. Mělo by být v zájmu vývojových týmů uvolňovat software včas dle časového harmonogramu, aby software byl dostatečně otestován a neodhalené defekty se nepřenášely do následujících softwarových smyček. Navrhované řešení spočívá v dodržování harmonogramu vývojářského týmu a tento požadavek by měl vyžadovat softwarový test manažer.

2. Návrh řešení je jako předchozí faktor těžko ovlivnitelný ze strany procesu plzeňského oddělení, jelikož se jedná o procesy zasahující do týmů specifikátorů softwaru, které komunikují se zákazníkem. Ze strany specifikátorů je kladen důraz na včasné obdržení zákaznických požadavků, aby získali dostatečnou časovou rezervu pro vytvoření softwarových požadavků. Hlavní problém se nachází u zákazníka, který nemá své požadavky ujasněné.

3. Návrh opatření lze ovlivnit častějšími a pravidelnými telekonferencemi dvakrát v týdnu realizované v ranních hodinách z důvodu časového posunu. V případě potřeby využívat pro komunikaci email a v akutních případech neplánované telekonference.

4. Návrh méně častého faktoru lze snížit kladením důrazu na softwarové test manažery, aby testovací případy byly včas delegovány do týmu plzeňského oddělení a tím nedocházelo ke zkrácení potřebného času k otestování. Dále je doporučeno se koordinátorovi dotazovat pravidelně každý týden na telefonních schůzkách na očekávané testovací případy a následující harmonogram testovacích prací.

Při vzniku situace, kdy koordinátor obdrží testovací balíku Job s nedostačenou časovou rezervou k otestování, je nutné, aby koordinátor apeloval na prodloužení termínu otestování. Pro zajištění dostatečně kvalitního otestování softwaru.

4.6.2 Navrhované řešení přípravy specifikace testování

Navrhovaná optimalizace přípravy specifikace testování bude rozdělena na dvě optimalizace. První optimalizace se bude zaměřovat čínskými projekty a druhá optimalizace se bude zabývat projekty Jaguar a Land Rover. Optimalizace u čínských projektů není zcela zásadní, jelikož nejsou shledány významné faktory, které by komplikovaly proces přípravy specifikace.

Návrh optimalizace procesů přípravy specifikace čínských projektů dle faktorů:

1. Návrh faktoru je sice méně zásadní, ale i tak je možné zvážit návrh optimalizace procesu. Navrhované řešení spočívá v důsledné analýze testovacích případů ze strany softwarového test manažera, který by neměl zadávat testovací případy oddělení, které nedisponuje potřebným hardwarovým vybavením nebo znalostmi. Důslednější analýza by měla probíhat i ze strany koordinátora týmu, jelikož může nastat situace, kdy softwarový test manažer deleguje testovací případ k otestování do týmu RoW a je tedy na koordinátorovi, aby takový testovací případ vrátil softwarovému test manažerovi k otestování do jiného oddělení, které disponuje potřebným vybavením a znalostmi.

2. Návrh faktoru je také méně významný a je dost podobný jako předchozí. Zde je návrh optimalizace také v důsledné analýze relevantnosti testovacího balíku Job obsahující jednotlivé testovací případy k testované variantě projektu ze strany softwarového test manažera a také ze strany koordinátora.

Návrh optimalizace procesů přípravy specifikace u projektů Jaguar a Land Rover dle faktorů:

1. Návrh je velice významný, jelikož počet nerelevantních testovací případů dosahuje hodnoty 14 %. Optimalizace tohoto faktoru je velmi složitá, jelikož se jedná o komplexní procesy, které zasahují mimo plzeňské oddělení a také mimo působnost softwarového test manažera, který deleguje testovací případy týmu plzeňské lokace. V tomto případě je nutné změnit proces již na straně specifikátorů, kteří by měli podávat přesné informace a specifikovat softwarovým test manažerům, které testovací případy jsou pro testovanou variantu relevantní. Analýza relevantnosti je

nutná v případech pro testovací případy vzniklé za účelem otestování nového modulu komponenty nebo na základě nalezeného defektu již používaného modulu.

2. Návrh je již méně významný a jeho optimalizace také spočívá v důsledné analýze relevantnosti testovacího balíku Job obsahující jednotlivé testovací případy k testované variantě projektu ze strany software test manažera a také ze strany koordinátora.

Výše uvedené návrhy optimalizace mají omezit zdlouhavé hledání relevantnosti varianty na straně testerů a tím docílit snížení časové náročnosti na otestování celého testovacího balíku Job a tím i snížením nákladů.

4.6.3 Navrhované řešení realizace manuálního komponentního testování

Optimalizace řešení realizace testování dle analyzovaných faktorů:

1. Optimalizace optimální vytíženosti HiLů je vhodná pro společnost, která se snaží HiLy maximálně využít, jelikož se jedná o velmi drahé zařízení s nákladným provozem vzhledem k licencím měřících nástrojů a softwaru, které jsou součástí HiLů. Optimalizace spočívá v důsledné kontrole rezervace nasazením monitorovacího nástroje, který bude sledovat vytížení HiLů a kontrolovat, zda je po dobu rezervace HiL opravdu využíván a není zbytečně blokován pro jiná měření. Uvolnění kapacity PHS HiLů pro plzeňské oddělení způsobeno obsazením PHS HiLů čínským oddělením je možné vyřešit omezením rezervace v určitý časový úsek. Časový posun čínského oddělení je oproti SEČ v letních měsících 6 hodin a v zimních měsících 7 hodin. Na základě časového posunu se nabízí možnost využívání HiLu PHS čínským oddělením od 1:00 do 8:00 hod SEČ a omezit rezervaci uživatelům jen v uvedeném časovém období. Dále se nabízí optimalizace cloudovým řešením. Vybrané HiLy by byly přepnuty do režimu Cloud. Tato možnost lze využít jen tehdy, pokud by plzeňské a čínské oddělení testovalo stejný model se stejnou softwarovou smyčkou. Tato možnost je výhodná především při automatickém testování, kdy tester ladí automatické testy dané komponenty v programu EXAM na své pracovní stanici a pak je odešle na cloudový HiL a testy se zařadí do fronty již před testy, které právě běží. Po dokončení testů si tester prohlédne vyhodnocení a měření testů na své lokální stanici v programu EXAM.

Výhodou je, že na Hilu může v jednu chvíli testovat více testerů zároveň, aniž by byli nuceni se na HiLu střídat. Nevýhodou je manuální měření, které je umožněno pouze jednomu testerovi a v té době je nutné HiL odpojit z režimu cloud.

2. Optimalizace využitelnosti jednotek s cílem počtu skutečného stavu používaných a nevyužívaných řídících jednotek spočívá v úpravě rezervace jednotek v rezervačním systému HiL, kdyby nebyla možná rezervace HiLu bez rezervace jednotky. V současné době se používá povinná položka pouze u PSP čísla, který slouží k účtování za použití HiLu. V současnosti by v rezervaci přibyla druhá povinná položka „řídící jednotka“ bez, které by nebylo možné rezervaci provést. Tímto způsobem by byl odstraněn problém s rezervacemi HiLů bez jednotek. Dále by uživatelé čínského oddělení byli omezeni počtem využití řídících jednotek v rezervaci z důvodu udržení testovací kapacity plzeňského oddělení.
   Počet dostupných jednotek k rezervaci pro čínské oddělení:
   - 8HP50: 2 kusy
   - 8HP3G: 2 kusy
   - 9HP50: 2 kusy

   Dalším řešením problému s kapacitou jednotek je možné docílit cloudovým režimem HiL, kdy je zapotřebí k testování nižší počet řídících jednotek než v běžném režimu.

3. Optimalizace nefunkčního modelu ControlDesk není přímo v procesu oddělení plzeňské lokace, jelikož nastavení modelů je v kompetenci modelářů, kteří společně s vývojáři softwaru vytvoří správné nastavení modelu k softwarové smyčce. Řešením je požadovat na softwarovém test manažerovi, aby zajistil vždy k testované softwarové smyčce aktuální nastavení modelu. Další povinností softwarového test manažera je ověřit, zda software, který je určen k testování softwarové smyčky je připraven pro testování. Optimalizace třetího faktoru zajistí zvýšení efektivity testera, jelikož nebude čas věnovat zkoumání zprovoznění modelu nebo nefunkčního softwaru.

4.6.4 Navrhované řešení zpracování výsledků manuálního komponentního testování

Optimalizace řešení zpracování výsledků testování na základě provedené analýzy popsané v kapitole 4.4.4 Analýza zpracování výsledku manuálního komponentního

testování. V analýze jsou uvedeny dva faktory, kde optimalizace řešení je pro oba faktory společná.

Optimalizace řešení obou faktorů současně spočívá v nasazení nástroje, který by umožnil splnit všechny funkce a možnosti nástrojů, které v současné době společnost využívá. Nástroj by měl umožňovat po napsaní testů a po jejich otestování přímo vložit výsledek do vytvořených testů. Testovací případ by byl rovnou přiřazen k vytvořeným testům. Dále by měl umožňovat nahrání naměřených dat, založení ticketu při odhalení defektu. Jednalo by se o komplexní správu testování, která by umožňovala exportovat pdf nebo xml soubory, které by obsahovaly všechny informace ohledně testovacího případu. Neodpadla by zřejmě nutnost ukládat naměřená data na pevný nebo verzovací disk na intranetu společně se souborem s informacemi ohledně testovacího případu pro potřeby pozdější archivace. Tento nástroj by zároveň vyřešil nedostatky nástrojů ClearQuest a DOORS, které jsou již v současné době zastaralé. Tento nástroj by umožnil komplexní a přehlednou správu testů nejenom pro testery při zpracování výsledků z měření, ale také pro koordinátora při zadávaní práce testerům. Dále také pro softwarového test manažera pro přehlednější řízení testování a diagnostického manažera pro snadnější analyzování ticketů, jako je např. nástroje Windchill RV&S Client nebo IBM Engineering Requirements Management DOORS Next.

4.7 Návrh opatření řešení procesu automatického regresního testování

V této kapitole budou navržená opatření optimalizace procesů automatického regresního testování na základě provedené analýzy uvedené v kapitole 4.5 Analýza současného procesu automatického regresního testování.

4.7.1 Navrhované řešení procesů automatického regresního testování

Návrh zlepšení dle jednotlivého faktoru:

Návrh řešení je zároveň stejný pro oba faktory. Spočívá v důkladné analýze relevantnosti a testovatelnosti variant modelů pro jednotlivé projekty ze strany software test manažera nebo specifikátora. Jedná se o proces, který není přímo v kompetenci oddělení, je tedy nutná kontrola validace testovacích případů na straně koordinátora týmu a nerelevantní a netestovatelné varianty reportovat software test manažerovi.

4.7.2 Navrhované řešení přípravy automatického regresního testování

Návrh zlepšení dle jednotlivého faktoru:

Návrh řešení faktoru je pro plzeňské oddělení těžko ovlivnitelný, jelikož se jedná o procesy, které jsou v kompetenci jiného vývojového týmu v odlišném oddělení, které sídlí v jiné geografické lokaci. Koordinátorovi v plzeňské lokaci je umožněno pouze apelovat na včasné dodání termínů testovacích případů k otestování prostřednictvím softwarového test manažera.

4.7.3 Navrhované řešení realizace automatického regresního testování

Návrh zlepšení dle jednotlivého faktoru:

Návrh řešení faktoru spočívá v navýšení kapacity technické podpory automatizačního programu EXAM, který zajišťuje přípravu jednotlivých operací, doplnění signálů a proměnných do databází jednotlivých modelů a projektů. Dále stojí za zvážení, které testovací případy se vyplatí automatizovat v poměru času potřebného k samotnému manuálnímu testování versus čas potřebný k automatizaci. Automatizace je výhodná u testovacích případů, u kterých se testování provádí opakovaně.

4.7.4 Navrhované řešení zpracování výsledků automatického regresního testování

Návrh zlepšení dle jednotlivých faktorů:

1. Návrh řešení prvního faktoru spočívá v realizaci kalibrační revize, která bude provedena před samotným automatickým regresním testováním. Kalibrační revize by měla odhalit nesprávné kalibrace v softwaru testované varianty modelu projektu. Cílem je snížit počet defektů nalezených v automatickém testování, který sníží časovou náročnost při vytváření a analyzování ticketů. Kalibrační revizi by měl provádět specifikátor softwarových požadavků. Implementace bude popsána v kapitole 5.1.1 Výsledky automatického regresního testování po kalibrační revizi.

2. Návrh řešení druhého faktoru se zabývá korekcí softwarových požadavků. Optimalizace má omezit rozdíly mezi softwarovými a zákaznickými požadavky a zajistit požadované funkce a chování softwaru dle požadavků zákazníka. Cílem je opět snížit počet defektů nalezených v automatickém testování, který sníží časovou náročnost při vytváření a analyzování ticketů. Korekci softwarových požadavků by měl provádět specifikátor softwarových požadavků. Implementace bude popsána

v kapitole 5.1.2 Výsledky automatického regresního testování po korekci softwarových požadavků.

3. Návrh optimalizace faktoru vysoké vytíženosti oficiální databáze spočívá ve vytvoření nové zrcadlové databáze, do které se nakopíruje kolekce z plzeňské databáze, která se pak zrcadlí na diskové pole serveru, na kterém je uložena oficiální databáze. Kolekce se pak prostřednictvím vzdálené plochy serveru zkopíruje do oficiální databáze. Optimalizací byl zkrácen čas exportu kolekce z řádu jednotek hodin na pouhé desítky minut.

5 Výsledky a diskuse

V této kapitole budou představeny dosažené výsledky implementace na základě provedené analýzy současných procesů a návrhů optimalizace procesů automatického regresního testování softwaru automatických převodovek osobních automobilů.

Procesy lze rozdělit na ty, které jsou ovlivnitelné pouze v plzeňském oddělení a následně na procesy, které se prolínají do ostatních oddělení, které se nacházejí v jiných geografických lokací. U těchto procesů by byla implementace optimalizace procesů velice složitá. Z tohoto důvodu byla provedena realizace optimalizace procesů, které jsou v kompetencích plzeňského oddělení. Jedná se o procesy automatického regresního testování v podobě implementace revizí kalibrací a korekcí softwarových požadavků, jehož cílem bylo ověřit snížení časové náročnosti v procesu automatického regresního testování. Na základě provedeného měření a obdržení výsledků, které jsou popsány v níže uvedené kapitole 5.1 Dosažené výsledky po implementaci automatického regresního testování, by bylo vhodné aplikovat kalibrační revize a korekci softwarových požadavků do standartních procesů automatického regresního testování. Dále je vhodné implementovat návrh optimalizace rezervačního systému HiL, který spočívá v omezení času rezervace od 2:00 do 8:00 SEČ pro čínské oddělení a dále pak zvolit výběr řídící jednotky jako povinnou položku při rezervaci. Tím bude docíleno uvolnění kapacity měření pro plzeňské oddělení.

Další návrh by spočíval v přepnutí jednoho zařízení HiL do režimu cloud, které by společně vyžívaly plzeňské a čínské oddělení. Na zařízení budou posílány a pouštěny automatické regresní testy společného testovaného softwaru a po měsíčním provozu na základě výsledků z využití zařízení by bylo na zvážení, zda tento návrh nezahrnout do standartních procesů. Další implementace již nebyly provedeny u dalších návrhů optimalizací, jelikož se jedná o procesy, kde jejich nasazení je složitější a projevené změny by bylo možné ověřit až v dlouhodobějším horizontu.

5.1 Dosažené výsledky po implementaci automatického regresního testování

5.1.1 Výsledky automatického regresního testování po kalibrační revizi

Na základě navrhované optimalizace procesu realizace automatického regresního testování byla provedena implementace v podobě nasazení kalibrační revize. Implementace byla provedena na třech typech softwarů automatických převodovek 8HP2G, 8HP3G

a 9HP50. Nejprve byly otestovány softwary před provedením kalibrační revize. V tabulce 1 Výsledky testů před kalibrační revizí jsou uvedeny jednotlivé výsledky testovaných softwarů. U softwaru 8HP2G bylo provedeno celkem 118 testů, z toho bylo 110 pozitivních a 8 negativních testů. U softwaru 8HP3G bylo provedeno celkem 413 testů, z toho bylo 351 pozitivních a 62 negativních testů. U posledního testovaného softwaru 9HP50 bylo provedeno celkem 444 testů, z toho bylo 331 pozitivních a 113 negativních testů. Poté byl stejný software otestován po kalibrační revizi. V tabulce 2 Výsledky testů po kalibrační revizi jsou uvedeny výsledky, z nichž je patrné na první pohled snížení počtu defektů. U softwaru 8HP2G bylo provedeno celkem 118 testů z toho bylo 116 pozitivních testů a 2 negativní testy. U softwaru 8HP3G bylo provedeno celkem 413 testů z toho bylo 388 pozitivních a 25 negativních testů. U posledního testovaného softwaru 9HP50 bylo provedeno celkem 444 testů z toho bylo 404 pozitivních a 40 negativních testů. Kalibrační revizí bylo docíleno snížení defektů v průměru o 66,43 %, dle výsledků jednotlivých softwarů uvedených v tabulce 3 Srovnání úbytku defektů po kalibrační revizi. Porovnání úbytku defektů po provedení kalibrační revize jednotlivých testovaných softwarů je zobrazeno na následujících grafech. V grafu 1 Porovnání úbytku defektů po kalibrační revizi u 8HP2G je patrný úbytek o šest defektů. Dále u grafu 2 Porovnání úbytku defektů po kalibrační revizi u 8HP3G je zřejmý úbytek v počtu 37 defektů. Následně u grafu 3 Porovnání úbytku defektů po kalibrační revizi u 9HP50 bylo docíleno snížení o 73 defektů. Nadále bylo docíleno snížení časové náročnosti při tvorbě ticketů. Průměrná doba na reportování defektu je 18 minut. V tabulce 4 Snížení časů defektů po kalibrační revizi jsou uvedeny výsledky, kdy bylo ušetřeno při testování třech softwarů v průměru 11,6 hodiny.

Tabulka 1 Výsledky testů před kalibrační revizí

Testy	8HP2G	8HP3G	9HP50
Pozitivní	110	351	331
Negativní	8	62	113
Celkem	118	413	444

Zdroj: vlastní zpracování, (2023)

Tabulka 2 Výsledky testů po kalibrační revizi

Testy	8HP2G	8HP3G	9HP50
Pozitivní	116	388	404
Negativní	2	25	40
Celkem	118	413	444

Zdroj: vlastní zpracování, (2023)

Tabulka 3 Srovnání úbytku defektů po kalibrační revizi

Testy	8HP2G	8HP3G	9HP50	Průměr
Úbytek negativních	6	37	73	38,67
Úbytek negativních v %	75,00	59,68	64,60	66,43

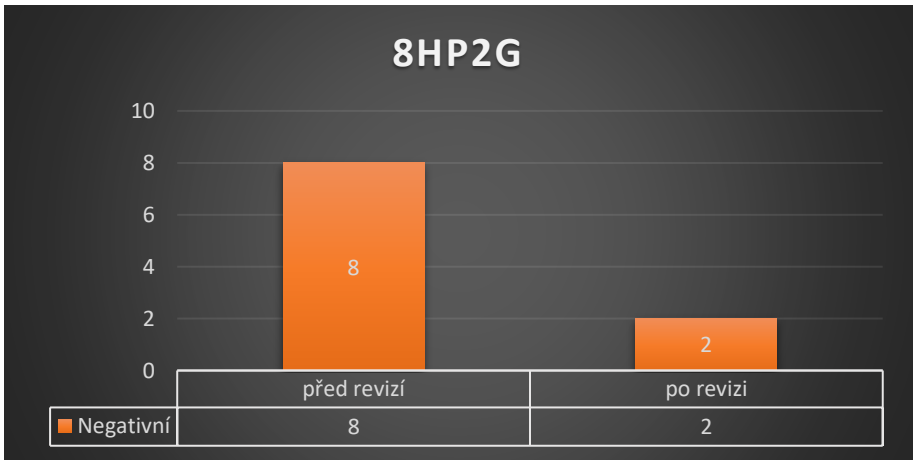
Zdroj: vlastní zpracování, (2023)

Tabulka 4 Snížení časů defektů po kalibrační revizi

Položky	8HP2G	8HP3G	9HP50	Průměr
Úbytek negat.	6	37	73	38,67
Ušetřený čas v min	108,00	666,00	1314,00	696,00
Ušetřený čas v hod	1,80	11,10	21,90	11,60

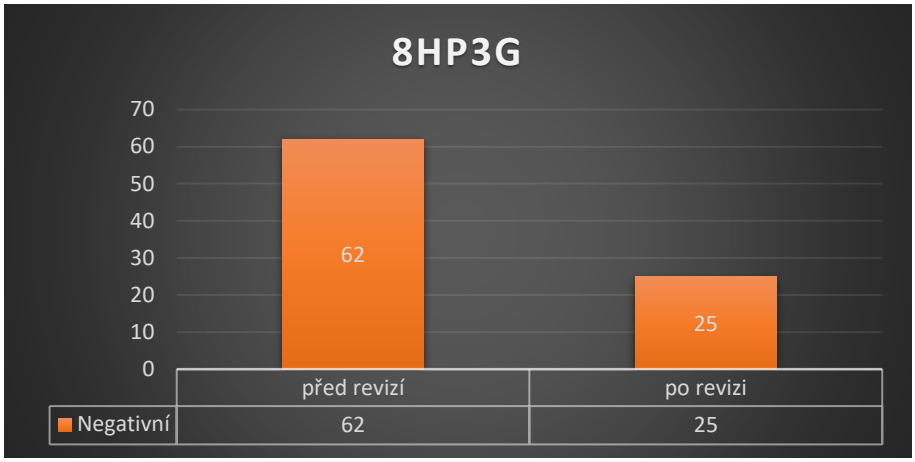
Zdroj: vlastní zpracování, (2023)

Graf 1 Porovnání úbytku defektů po kalibrační revizi u 8HP2G



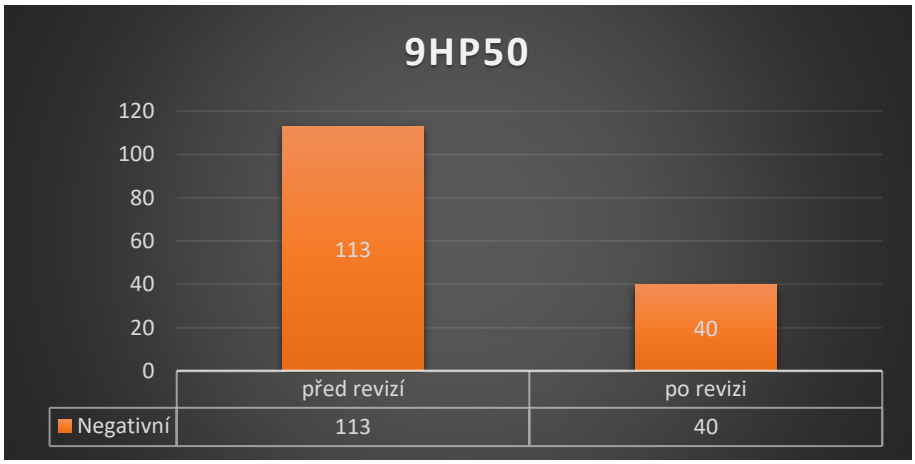
Zdroj: vlastní zpracování

Graf 2 Porovnání úbytku defektů po kalibrační revizi u 8HP3G



Zdroj: vlastní zpracování

Graf 3 Porovnání úbytku defektů po kalibrační revizi u 9HP50



Zdroj: vlastní zpracování

5.1.2 Výsledky automatického regresního testování po korekci softwarových požadavků

Na základě navrhované optimalizace procesu realizace automatického regresního testování byla provedena implementace korekce softwarových požadavků. Implementace byla provedena na třech typech softwarů automatických převodovek 8HP2G, 8HP3G a 9HP50. Nejprve byly otestovány softwary před provedením korekce softwarových požadavků. V tabulce 5 Výsledky testů před korekcí softwarových požadavků jsou uvedeny jednotlivé výsledky testovaných softwarů. U softwaru 8HP2G bylo provedeno celkem 118 testů, z toho bylo 110 pozitivních a 8 negativních testů. U softwaru 8HP3G bylo provedeno celkem 413 testů, z toho bylo 351 pozitivních a 62 negativních testů. U posledního

testovaného softwaru 9HP50 bylo provedeno celkem 444 testů, z toho bylo 331 pozitivních a 113 negativních testů. Poté byl stejný software otestován po kalibrační revizi. V tabulce 6 Výsledky testů po korekci softwarových požadavků jsou uvedeny výsledky, z nichž je patrné na první pohled snížení počtu defektů. U softwaru 8HP2G bylo provedeno celkem 118 testů, z toho bylo 112 pozitivních a 6 negativních testů. U softwaru 8HP3G bylo provedeno celkem 413 testů, z toho bylo 376 pozitivních a 41 negativních testů. U posledního testovaného softwaru 9HP50 bylo provedeno celkem 444 testů z toho bylo 371 pozitivních a 73 negativních testů. Korekcí softwarových požadavků bylo docíleno snížení defektů v průměru o 31,42 %, dle výsledků jednotlivých softwarů uvedených v tabulce 7 Srovnání úbytku defektů po korekci softwarových požadavků. Porovnání úbytku defektů po provedení korekce softwarových požadavků jednotlivých testovaných softwarů je zobrazeno na následujících grafech. Na grafu 4 Porovnání úbytku defektů po korekci softwarových požadavků u 8HP2G, kde je patrný úbytek o dva defekty. Dále u grafu 5 Porovnání úbytku defektů po korekci softwarových požadavků u 8HP3G, je zřejmý úbytek v počtu 21 defektů. Následně u grafu 6 Porovnání úbytku defektů po korekci softwarových požadavků u 9HP50, bylo docíleno snížení o 40 defektů. Dále bylo docíleno snížení časové náročnosti při tvorbě ticketů. Výsledky jsou uvedeny v tabulce 8 Snížení časů defektů po korekci softwarových požadavků, kdy bylo ušetřeno při testování třech softwarů v průměru 6,3 hodiny.

Tabulka 5 Výsledky testů před korekcí softwarových požadavků

Testy	8HP2G	8HP3G	9HP50
Pozitivní	110	351	331
Negativní	8	62	113
Celkem	118	413	444

Zdroj: vlastní zpracování, (2023)

Tabulka 6 Výsledky testů po korekci softwarových požadavků

Testy	8HP2G	8HP3G	9HP50
Pozitivní	112	376	371
Negativní	6	41	73
Celkem	118	417	444

Zdroj: vlastní zpracování, (2023)

Tabulka 7 Srovnání úbytku defektů po korekci softwarových požadavků

Testy	8HP2G	8HP3G	9HP50	Průměr
Úbytek negat.	2	21	40	21
Úbytek negat. v %	25,00	33,87	35,40	31,42

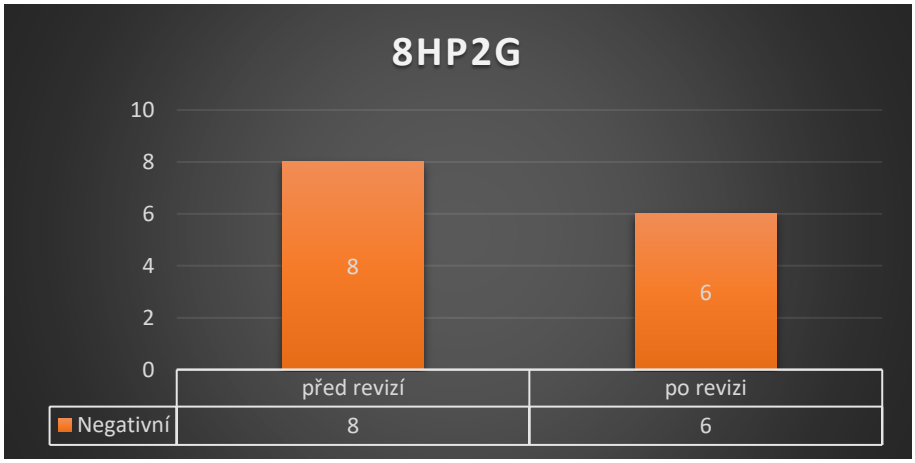
Zdroj: vlastní zpracování, (2023)

Tabulka 8 Snížení časů defektů po korekci softwarových požadavků

Položky	8HP2G	8HP3G	9HP50	Průměr
Úbytek negat.	2	21	40	21,00
Ušetřený čas v min	36,00	378,00	720,00	378,00
Ušetřený čas v hod	0,60	6,30	12,00	6,30

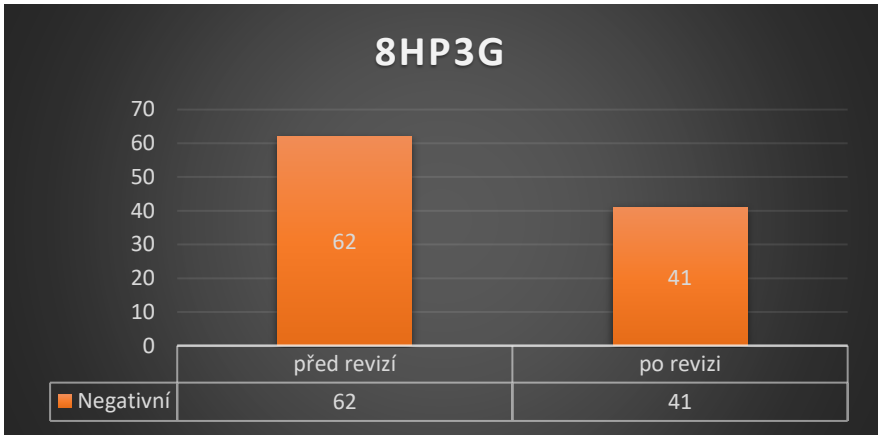
Zdroj: vlastní zpracování, (2023)

Graf 4 Porovnání úbytku defektů po korekci softwarových požadavků u 8HP2G



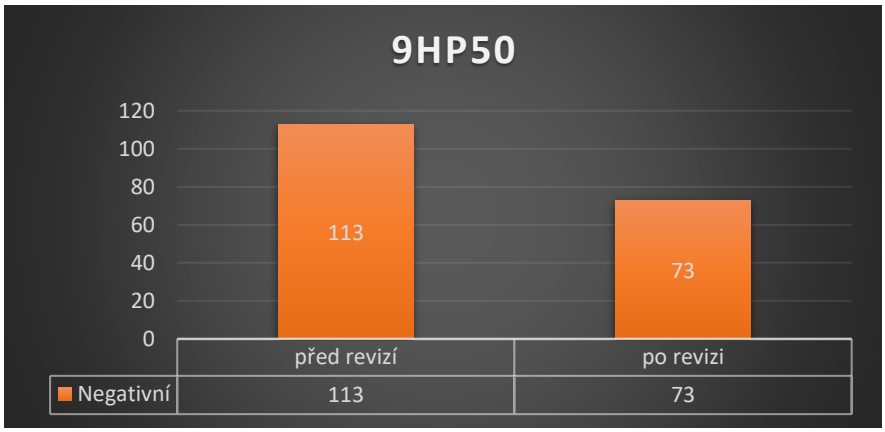
Zdroj: vlastní zpracování, (2023)

Graf 5 Porovnání úbytku defektů po korekci softwarových požadavků u 8HP3G



Zdroj: vlastní zpracování, (2023)

Graf 6 Porovnání úbytku defektů po korekci softwarových požadavků u 9HP50



Zdroj: vlastní zpracování, (2023)

5.2 Porovnání výsledků implementace

Jak je patrné z výsledků uvedených v kapitole 5.1 Dosažené výsledky po implementaci automatického regresního testování a porovnání mezi tabulkou 3 Srovnání úbytku defektů po kalibrační revizi a tabulkou 7 Srovnání úbytku defektů po korekci softwarových požadavků, kalibrační revize má vyšší vliv na snížení defektů, kde bylo dosaženo snížení v průměru o 66,43 %, na rozdíl od korekce softwarových požadavků, kde bylo dosaženo snížení v průměru 31,42 %. Výhodou kalibrační revize je také to, že nevyžaduje velkou časovou náročnost, která činí přibližně 8 hodin až 16 hodin a je závislá na počtu kalibrací vyskytujících se v aktuálním softwaru. Na rozdíl korekce systémových požadavků je časově daleko náročnější, pohybuje se v rozmezí 20 hodin až 30 hodin v závislosti na počtu systémových a zákaznických požadavků.

6 Závěr

První část práce se věnuje základním pojmům potřebných k pochopení problematiky testování softwaru. V úvodní části jsou popsány tradiční a agilní metodiky vývoje softwaru. Teoretická část se následně zabývá jednotlivými principy, procesy a technikami testování softwaru. V závěru teoretické části jsou uvedeny procesy spojené s managmentem testování, který je nezbytný pro plánování a řízení testování.

Po teoretické části následuje praktická část, jejíž hlavním cílem je analýza současných procesů testování a návrh optimalizace procesů testování v oblasti testování softwaru automatických převodovek do osobních automobilů ve společnosti ZF Engineering Plzeň. Po představení společnosti byly popsány současné procesy testování softwaru, vysvětlena funkce automatické převodovky pro osobní automobily, představeny nástroje a hardware využívaný pro testování softwaru. První krok popisuje aktuální stav procesů testování softwaru techniky manuálního komponentního testování a automatického regresního testování ve sledovaném testovacím oddělení. Na základě aktuálních procesů byla provedena analýza jednotlivých procesů, ze kterých byly definované a popsané jednotlivé klíčové faktory, které negativně ovlivňují procesy manuálního komponentního a automatického regresního testování. Po provedené analýze byla navrhnuta a popsána jednotlivá opatření návrhu řešení, jejichž cílem je odstranit nebo snížit analyzované faktory.

V závěru diplomové práce vyplývá rozdělení návrhů na optimalizace, které je možné realizovat okamžitě; a na opatření, která k realizaci vyžadují dlouhodobé sledování pro ověření navrhovaných řešení. Pro implementaci je rozhodující navrhované řešení procesů, zdali se jedná o procesy zasahující i mimo oddělení vybrané společnosti či nikoliv. Z výsledků měření bylo zjištěno snížení defektů, v případě provedení kalibračních revizí v průměru o 66,43 % a v případě korekce softwarových požadavků v průměru o 31,42 %. Výsledky měření z provedené implementace poskytují potřebné údaje k ověření navrhovaných opatření a mohou dále posloužit jako podklady pro realizaci nasazení navrhovaných opatření do standartních procesů manuálního komponentního a automatického regresního testování. Stojí za zvážení, zda navrhované řešení procesů automatického regresního testovaní nezahrnout do standartních procesů, jelikož má pozitivní vliv na snížení časové náročnosti testování, v případě kalibračních revizí v průměru o 11,6 hodin a v případě korekce softwarových požadavků snížení v průměru o 6,3 hodiny. Tím je docíleno ušetření nákladů při zachování kvality testování softwaru.

7 Seznam použitých zdrojů

(1) ISTQB. *Certified Tester Foundation Level Syllabus Version 2018 V3.1*. ISTQB, 2018.

(2) BUREŠ, Miroslav,. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016.

(3) PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002.

(4) CHARVAT, Jason. *Project management methodologics: selecting, implementing and supporting methodologies and processes for project*. New Jersey: John Wiley, 2003.

(5) STOICA, Marian, Marinela MIRCEA a Bogdan GHILIC-MICU. Software Development: Agile vs. Traditional. *Informatica Economica*. 2013, **17**(42013), 64-76. ISSN 14531305. Dostupné z: doi:10.12948/issn14531305/17.4.2013.06

(6) *SDLC Tutorial: SDLC - Waterfall Model* [online]. tutorialspoint, 2023 [cit. 2023-03-19]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

(7) DVOŘÁK, D.,. *Project portfolio management*. Brno: Computer Press, 2017.

(8) *Software Engineering | Spiral Model* [online]. A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305: GeeksforGeeks, 2023 [cit. 2023-03-19]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-spiral-model/

(9) HAMILTON, Thomas. *V-Model in Software Testing* [online]. 4023 Kennett Pike #50286, Wilmington, Delaware, USA: Guru99, 2023 [cit. 2023-03-19]. Dostupné z: https://www.guru99.com/v-model-software-testing.html

(10) *What is V-model and W-model in Software Testing* [online]. 65 Broadway Suite 1101, New York NY 10006: Testbytes, 2017 [cit. 2023-03-19]. Dostupné z: https://www.testbytes.net/blog/v-model-and-w-model-software-testing/

(11) *What is agile methodology?* [online]. Red Hat, 2022 [cit. 2023-03-19]. Dostupné z: https://www.redhat.com/en/devops/what-is-agile-methodology

(12) MYSLÍN, Josef. *Procesy vývoje softwaru*. Praha: Vysoká škola podnikání a práva a.s., 2016.

(13) *Manifest Agilního vývoje software*. Dostupné také z: https://agilemanifesto.org/iso/cs/manifesto.html

(14) *Scrum Testing: A Detailed Guide to Testing on an Agile Team* [online]. Testim, 2019 [cit. 2023-03-19]. Dostupné z: https://www.testim.io/blog/scrum-testing-guide/

(15) *Scrum.org* [online]. Scrum.org, 2023 [cit. 2023-03-19]. Dostupné z: https://www.scrum.org/

(16) VOGELZANG, Johannes, Wilfried F. ADMIRAAL a Jan H. VAN DRIEL. Scrum Methodology as an Effective Scaffold to Promote Students' Learning and Motivation in Context-based Secondary Chemistry Education. *EURASIA Journal of Mathematics, Science and Technology Education*. 2019, **15**(12). ISSN 13058223. Dostupné z: doi:10.29333/ejmste/109941

(17) MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016.

(18) *Extreme Programming: Values, Principles, and Practices* [online]. Atlanta: AltexSoft, 2021 [cit. 2023-03-19]. Dostupné z: https://www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/

(19) *Feature Driven Development* [online]. ProductPlan, 2023 [cit. 2023-03-19]. Dostupné z: https://www.productplan.com/glossary/feature-driven-development/

(20) IBM Garage Methodology. In: *IBM* [online]. [cit. 2023-03-19]. Dostupné z: https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/

(21) *Software Testing Objectives* [online]. ToolsQA.com, 2021 [cit. 2023-03-19]. Dostupné z: https://www.toolsqa.com/software-testing/istqb/software-testing-objectives/

(22) *7 Principles of Software Testing* [online]. Functionize, 2022 [cit. 2023-03-19]. Dostupné z: https://www.functionize.com/blog/7-principles-of-software-testing

(23) *15 testing methods all developers should know* [online]. Circle Internet Services, 2020 [cit. 2023-03-19]. Dostupné z: https://circleci.com/blog/testing-methods-all-developers-should-know/?utm_source=google&utm_medium=sem&utm_campaign=sem-google-dg--emea-en-dsa-tROAS-auth-nb&utm_term=g_-_c__dsa_&utm_content=&gclid=CjwKCAjw5dqgBhBNEiwA7PryaC4LIP6bItQl4KXgU0yyQXY8YaW6CYUSIMM2jPHxWlZ4d9iABedEYxoCKhYQAvD_BwE

(24) *The Psychology Of Testing* [online]. ProfessionalQA.com, 2018 [cit. 2023-03-19]. Dostupné z: https://professionalqa.com/the-psychology-of-testing

(25) PITTET, Sten. *The different types of software testing* [online]. Atlassian, 2023 [cit. 2023-03-19]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

(26) MEARS, Nick. *Types of Software Testing: Functional & Non-Functional* [online]. Microfocus, 2020 [cit. 2023-03-19]. Dostupné z: https://community.microfocus.com/adtd/b/sws-alm/posts/types-of-software-testing-functional-non-functional

(27) *Change-Related Testing* [online]. ProgramsBuzz, 2020 [cit. 2023-03-19]. Dostupné z: https://www.programsbuzz.com/article/change-related-testing

(28) DEWHURST, Ryan. *Static Code Analysis* [online]. OWASP Foundation, 2023 [cit. 2023-03-19]. Dostupné z: https://owasp.org/www-community/controls/Static_Code_Analysis

(29) BLACK, Rex. *Managing the testing process: practical tools and techniques for managing software and hardware testing*. 3rd ed. Indianapolis, MN: Wiley, 2009. ISBN 9780470404157.

(30) GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL. *Foundations of software testing: ISTQB certification*. Fourth edition. Andover, Hampshire: Cengage, 2020. ISBN 978-1-4737-6479-8.

(31) BLACK, Rex a Jamie MITCHELL. *Advanced software testing: guide to the ISTQB advanced certification as an advanced technical test analyst*. 1st ed. Santa Barbara, CA: Rocky Nook, 2011. ISBN 978-1-933952-39-0.

(32) BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní*

testování softwaru: klíčové otázky pro efektivitu testovacího procesu. První vydání. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

(33) BOSE, Shreya. *Test Planning: A Detailed Guide* [online]. BrowserStack, 2022 [cit. 2023-03-19]. Dostupné z: https://www.browserstack.com/guide/test-planning

(34) *What Is Test Monitoring And Test Control?* [online]. Software Testing Help, 2023 [cit. 2023-03-19]. Dostupné z: https://www.softwaretestinghelp.com/test-monitoring-and-test-control/

(35) BATHRICK, Mark. *The Ultimate Guide to Risk Management in Testing* [online]. Virginia: Syntax, 2022 [cit. 2023-03-19]. Dostupné z: https://www.syntaxtechs.com/blog/the-ultimate-guide-to-risk-management-in-testing

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

Obrázek 1 Rámec vodopádového řízení projektu 17
Obrázek 2 Schéma spirálového modelu životního cyklu softwaru 18
Obrázek 3 Budova plzeňské pobočky ZF Engineering Plzeň s.r.o. 63
Obrázek 4 Schéma automatické převodovky 64
Obrázek 5 Řídící jednotka automatické převodovky 64
Obrázek 6 Softwarová struktura řídící jednotky automatické převodovky 65
Obrázek 7 W-model ZF 66
Obrázek 8 Prostředí ClearQuest 69
Obrázek 9 Náhled nástroje DOORS 70
Obrázek 10 Náhled nástroje Omnitracker 71
Obrázek 11 HiL Scalexio 72
Obrázek 12 Prostředí ControlDesk 73
Obrázek 13 Prostředí programu INCA 74
Obrázek 14 Nástroj automatického testování EXAM 75
Obrázek 15 Schéma Vodopádového modelu životního cyklu softwaru 112
Obrázek 16 Schéma V-modelu životního cyklu softwaru 112
Obrázek 17 Schéma W-modelu životního cyklu softwaru 113
Obrázek 18 Schéma životního cyklu metodiky SCRUM 113
Obrázek 19 Schéma struktury testování černé skříňky 114
Obrázek 20 Schéma struktury testování bílé skříňky 114
Obrázek 21 Schéma životního cyklu vývoje defektu 115
Obrázek 22 Komunikační matice pro správu defektů v projektu 115

8.2 Seznam tabulek

Tabulka 1 Výsledky testů před kalibrační revizí 100
Tabulka 2 Výsledky testů po kalibrační revizi 101
Tabulka 3 Srovnání úbytku defektů po kalibrační revizi 101
Tabulka 4 Snížení časů defektů po kalibrační revizi 101
Tabulka 5 Výsledky testů před korekcí softwarových požadavků 103
Tabulka 6 Výsledky testů po korekci softwarových požadavků 103
Tabulka 7 Srovnání úbytku defektů po korekci softwarových požadavků 104
Tabulka 8 Snížení časů defektů po korekci softwarových požadavků 104

8.3 Seznam grafů

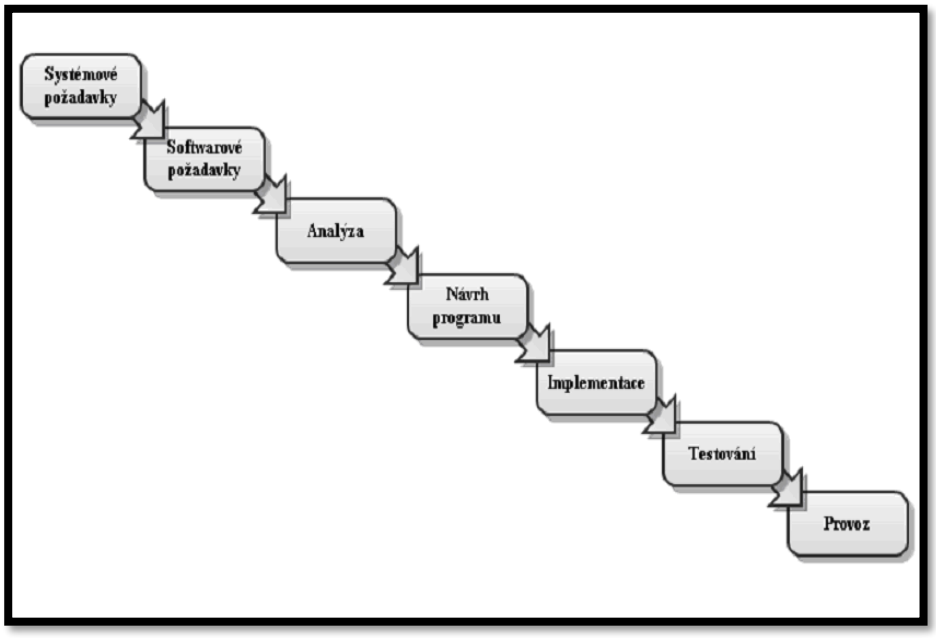
Graf 1 Porovnání úbytku defektů po kalibrační revizi u 8HP2G 101
Graf 2 Porovnání úbytku defektů po kalibrační revizi u 8HP3G 102
Graf 3 Porovnání úbytku defektů po kalibrační revizi u 9HP50 102
Graf 4 Porovnání úbytku defektů po korekci softwarových požadavků u 8HP2G 104
Graf 5 Porovnání úbytku defektů po korekci softwarových požadavků u 8HP3G 105
Graf 6 Porovnání úbytku defektů po korekci softwarových požadavků u 9HP50 105

8.4 Seznam použitých zkratek

A2L	Data Description for ECU Access
ASIS	Adaptive Shift Strategy
ATSYS	Shifting Control Software Component
BBF	Build By Feature
BMW	Bayerische Motoren Werke
BoB	Breakout box
CAN	Controller Area Network
CAN FD	CAN with Flexible Data rate
CDD	Complex Device Driver
CR	Change Request
DAS	Driver Alert Support
DTI	Driver Transmission Interface
EH	Error Handler
EXAM	EXtended Automation Method
FDD	Feature Driven Development
FMA	Failure Mode and Effect Analyses
HiL	Hardware in the Loop
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
PRAM	Pragmatic Risk Analyses and Management
PC	Personal computer
PDF	Portable Document Format
PSP	Projekt Struktur Plan
RCR	Requirement Change Request
RoW	Rest of World
RUP	Rational Unified Process
SABAL	Basic Signal Processing Application Layer
SABBL	Basic Signal Processing
SAP	Systeme Anwendungen Produkte
UML	Unified Modeling Language
XML	Extensible Markup Language

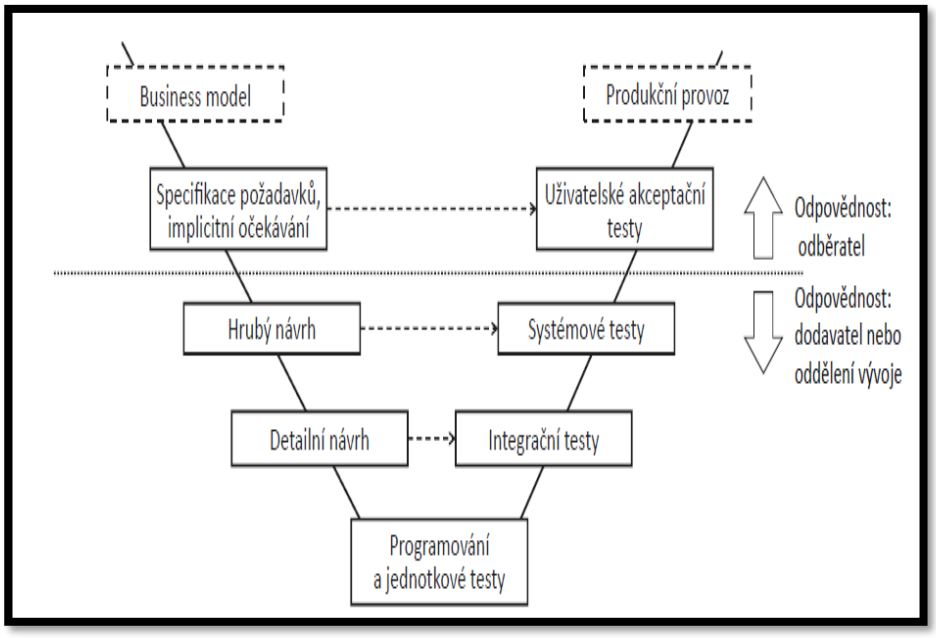
Příloha A

Obrázek 15 Schéma Vodopádového modelu životního cyklu softwaru



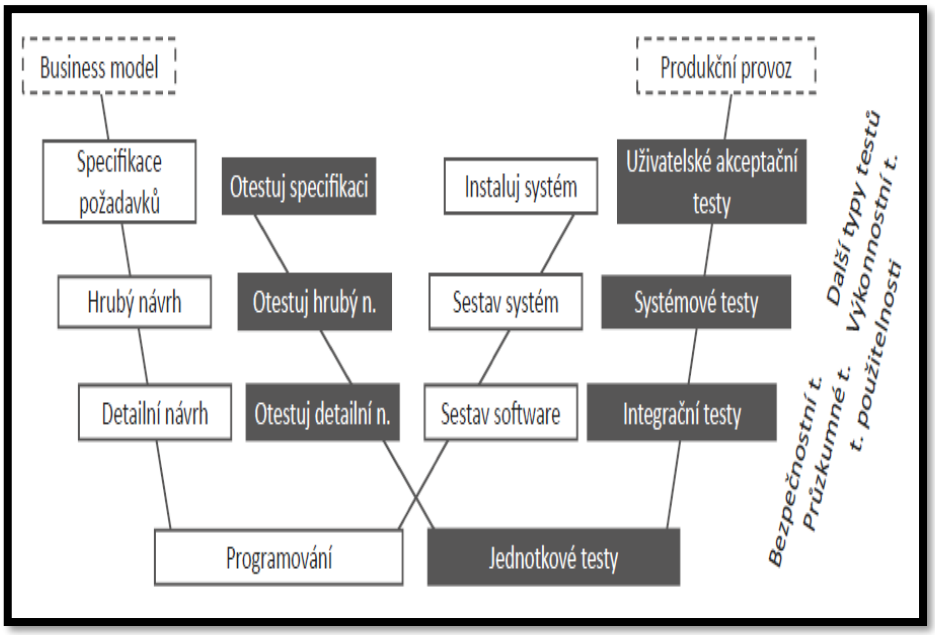
Zdroj: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/vodopadovy-model

Obrázek 16 Schéma V-modelu životního cyklu softwaru



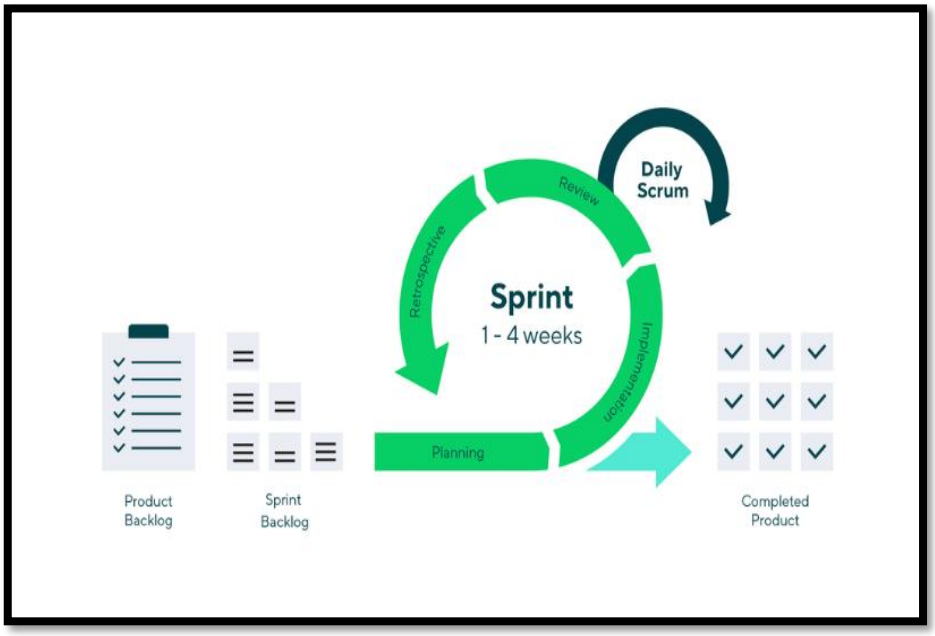
Zdroj: Bureš (2016) s. 28, obr. 2.2.

Obrázek 17 Schéma W-modelu životního cyklu softwaru



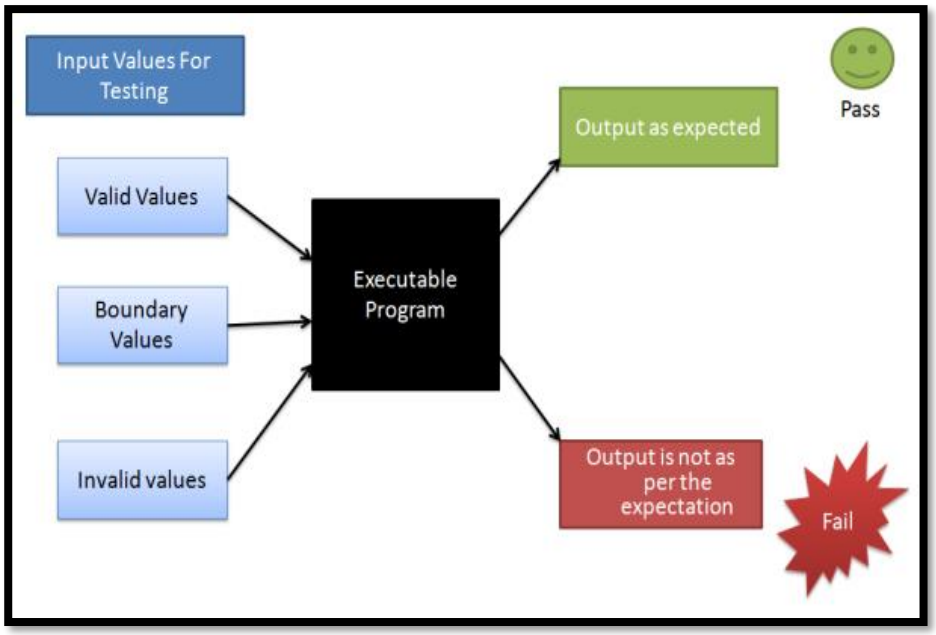
Zdroj: Bureš (2016) s. 29, obr. 2.3.

Obrázek 18 Schéma životního cyklu metodiky SCRUM



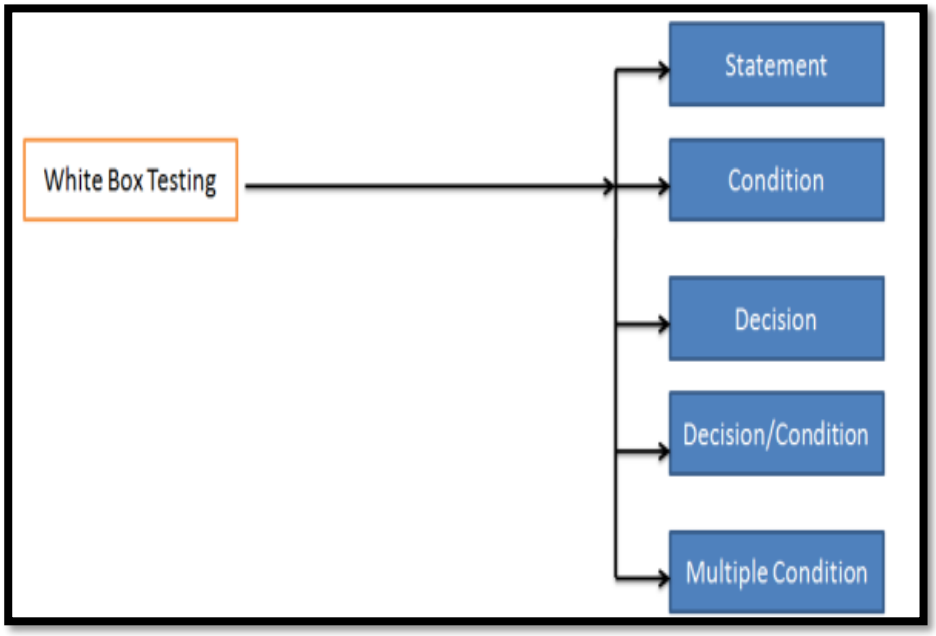
Zdroj: https://www.wrike.com/scrum-guide/scrum-sprints/

Obrázek 19 Schéma struktury testování černé skříňky



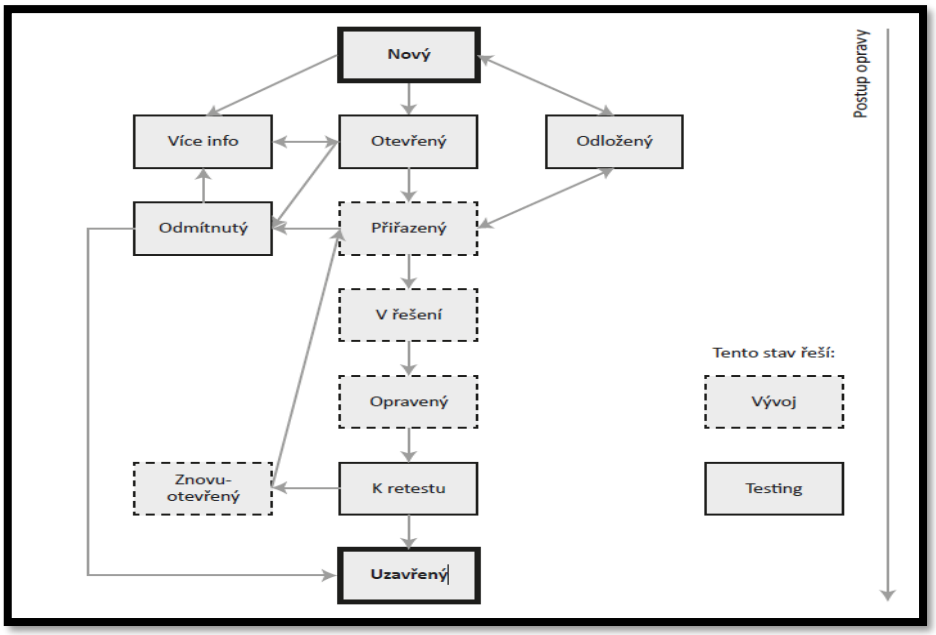
Zdroj: https://www.test-institute.org/What_Is_Software_Testing.php

Obrázek 20 Schéma struktury testování bílé skříňky



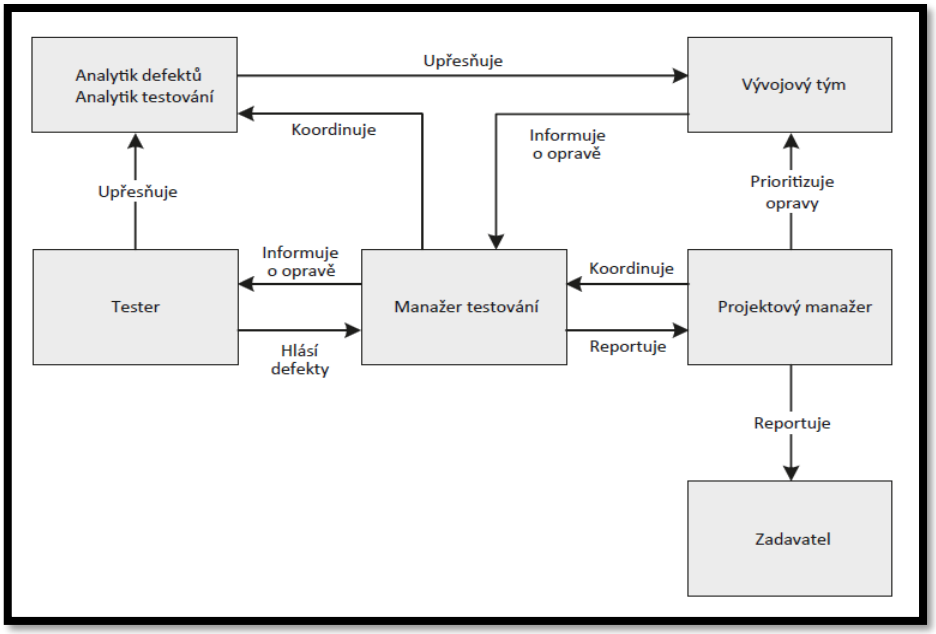
Zdroj: https://www.test-institute.org/What_Is_Software_Testing.php

Obrázek 21 Schéma životního cyklu vývoje defektu



Zdroj: Bureš (2016) s. 153, obr. 10.1.

Obrázek 22 Komunikační matice pro správu defektů v projektu



Zdroj: Bureš (2016) s. 154, obr. 10.2.