**ZÁPADOČESKÁ UNIVERZITA V PLZNI**
**FAKULTA ELEKTROTECHNICKÁ**

**KATEDRA TECHNOLOGIÍ A MĚŘENÍ**

# DIPLOMOVÁ PRÁCE

## Analýza procesního řízení

**Bc. Eva Andrlová** **2014**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
**Fakulta elektrotechnická**
Akademický rok: **2013/2014**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Eva ANDRLOVÁ** |
| Osobní číslo: | **E12N0001P** |
| Studijní program: | **N2612 Elektrotechnika a informatika** |
| Studijní obor: | **Komerční elektrotechnika** |
| Název tématu: | **Analýza procesního řízení** |
| Zadávající katedra: | **Katedra technologií a měření** |

Zásady pro vypracování:

1. Popište nástroje a metody pro optimalizaci procesů
2. Analyzujte vybraný nástroj pro popis a řízení procesů
3. Na aktuálním projektu popište postup procesního řízení
4. Zhodnoťte přínos využitých nástrojů na optimální průběh procesů

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

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

Seznam odborné literatury:


1. **KEŘKOVSKÝ, M.: Moderní přístupy k řízení výroby. Praze: C. H. Beck, 2009. ISBN 978-80-7400-119-2**
2. **PRESSMAN, S., R.: Software Engineering: A Practitioner's Approach. The McGraw-Hill Companies, 2004. ISBN 978-0072853186**
3. **FORSBERG, K., MOOZ, H., COTTERMAN, H.: Visualizing Project Management, 3rd edition, John Wiley and Sons, New York, NY, 2005. ISBN 242-248, 341-360**
4. **HOFFMAN, M., BEAUMONT, T.: Application Development: Managing the Project Life Cycle, Mc Press, 1997. ISBN 978-1883884451**
5. **Internetové zdroje**


Vedoucí diplomové práce: **Ing. Tomáš Řeřicha, Ph.D.**
Katedra technologií a měření

Datum zadání diplomové práce: **14. října 2013**

Termín odevzdání diplomové práce: **12. května 2014**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Ing. Vlastimil Skočil, CSc.
vedoucí katedry

V Plzni dne 14. října 2013

# Abstrakt

Předkládaná diplomová práce se zabývá typy modelů životních cyklů vývoje softwaru. Důraz je především kladen na V-model, jehož využití je ukázáno i v praxi. Jsou popsány všechny procesy, které daný V-model obsahuje. Na jejich základě je ukázáno procesní řízení v konkrétním projektu a jsou navrženy způsoby sledování výkonnosti vybraných procesů. Dále je v práci analyzováno, zda transformace stávajícího V-modelu do navrhovaného W-modelu povede k optimalizaci vývoje softwaru. Z hlediska vlivu na vyšší efektivitu vývoje jsou také porovnány nástroje IBM Rational Doors a Microsoft Excel.

## Klíčová slova

Softwarový vývoj, optimalizace, procesní řízení, vodopádový model, V-model, W-model.

# Abstract

This master thesis presents models of software development life cycles. The emphasis is placed mainly on the V-model and it´s usage is shown in practice. All processes contained in the V-model are described. On the basis of this information the process management in the specific project is demonstrated and ways of how to monitor the performance of the selected processes are proposed. The thesis also analyses whether the transformation of the original V-model into the suggested W-model leads to the optimization of software development. In terms of impact on higher development efficiency the comparison between the tool IBM Rational Doors and the tool Microsoft Excel is done.

## Key words

Software development, optimization, process management, waterfall model, V-model, W-model.

# Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

.....

podpis

V Plzni dne 12.5.2014 Eva Andrlová

# Poděkování

Tímto bych ráda poděkovala konzultantce Ing. Petře Fantové za profesionální rady a připomínky při zpracovávání práce a za ochotu a trpělivost při pravidelných konzultacích. Dále bych chtěla poděkovat vedoucímu diplomové práce Ing. Tomášovi Řeřichovi, Ph.D. za odborné vedení práce, cenné rady a komentáře.

# Obsah

# Seznam zkratek a symbolů

ARP.....Author Reader Principle

CAN.....Controller Area Network

CPU.....Central Processing Unit

CR.....Change Request

CQ.....ClearQuest

DM.....Department Manager

DOORS.....Dynamic Object Oriented Requirement System

DQ.....Department Quality Representative

HiL.....Hardware in the Loop

ID.....Identification

IEEE.....Institute of Electrical and Electronics Engineer

ISO.....lnternational Organization for Standardization

MiL.....Model in the Loop

MISRA-C.....Motor Industry Software Reliability Association C standard

OD.....Orderer

OEM.....Original Equipment Manufacturer

PMI.....Project Management Institute

PT.....Project Team members

RM Plan.....Requirement Management Plan

RAM.....Random Access Memory

REV.....Reviewer

ROM.....Read Only Memory

SIL.....Safety Integrity Level

Sil.....Software in the Loop

SPICE.....Software Process Improvement and Capability Determination

SPL.....Supplier

SRS.....Software Requirements Specification

SW.....Software

SWC..... Software Coordinator

SWCMR..... Software Configuration Management Representative

SWD..... Software Developer

SWFD..... Software Function Developer

SWI..... Software Integrator

SWAD..... Software Architecture Designer

SWPQ..... Software Project Quality Manager

SWT..... Software Tester

SWTM..... Software Test Manager

TBD.....To Be Determined

TBT.....Torsion Bar Torque

TL..... Team Leader

UML.....Unified Modeling Language

VBA.....Visual Basic for Application

VOB.....Versioned Object Base

WT.....Walkthrough

ZF FRD DTEC Develompent of electronic landing gear system for division C in ZF Friedrichshafen

ZF LS.....ZF Lenksysteme

ZF PLZ.....ZF Plzeň

ZF PLZ EDS.....ZF Plzeň Electronic Department - Software

ZF PLZ EDT.....ZF Plzeň Electronic Department - Testing

# Úvod

V současné době narůstá počet softwarových projektů ve všech průmyslových odvětvích. Na vývoj softwarových systémů jsou kladeny stále vyšší nároky z hlediska kvality řízení projektu a také optimalizace jednotlivých vývojových procesů. Řízení projektu vývoje softwarového produktu není ovšem jednoduchou záležitostí. Pokud řízení projektu není prováděno sofistikovaným způsobem, je velká pravděpodobnost, že projekt skončí neúspěšně. Kvůli zvýšení pravděpodobnosti úspěchu projektu bylo postupně vytvořeno několik modelů životního cyklu softwarového vývoje.

Práce je rozdělena do pěti hlavních části. V první kapitole je představeno šest typů modelů životního cyklu vývoje softwaru. Každý model je základně charakterizován i s identifikací kladů a záporů. Následující kapitola se zabývá V-modelem a jeho využitím v praxi, konkrétně ve společnosti ZF Engineering Plzeň, s.r.o. Společnost je nejprve stručně prezentována. Poté jsou popsány procesy dle V-modelu v oddělení elektroniky včetně procesů, které jsou vykonávány obchodním partnerem. V práci jsou zahrnuty všechny procesy kvůli jejich plnému pochopení a návaznosti a vazeb mezi nimi. Třetí kapitola se soustředí na seznámení s nástrojem pro popis procesů „Stages“. Je ukázáno jeho základní použití s jeho možnostmi využití. Pro lepší představu je popis nástroje doprovázen přiloženými obrázky, které zachycují jeho pracovní prostředí. Kapitola je zakončena určením výhod a nevýhod nástroje. Další část práce se zaměřuje na procesní řízení v projektu Honda a popisuje praktickou realizaci pracovních kroků ze dvou zvolených procesů „Analýza softwarových požadavků“ a „Softwarové testování“ v ZF V-modelu. Poslední kapitola se věnuje transformaci ZF V-modelu do ZF W-modelu. U procesů, které byly zvoleny v návaznosti na předchozí kapitolu, jsou navrženy vývojové diagramy. Oba typy modelů jsou porovnány a je analyzováno, zda provádění procesů ve společnosti ZF Engineering Plzeň, s.r.o. dle W-modelu povede ke zvýšení optimálnosti. Dále se pozornost soustředí na porovnání nástrojů IBM Rational DOORS a Microsoft Excel a určení, který z nich je vhodnější pro optimálnost a efektivitu procesu „Softwarové testování“.

# 1 Procesy v softwarovém inženýrství

Optimalizaci procesů v softwarovém inženýrství ovlivňuje několik faktorů - zvolený procesní model, definice pracovních kroků jednotlivých procesů a vybrané nástroje k jejich realizaci. Dalším faktorem mohou být pracovníci zodpovědní za plnění svých úkolů. Jsou dostatečně zkušení? Mají potřebné proškolení? Čím vhodněji jsou zmiňované faktory podchyceny, tím lze dosáhnout optimálnějšího průběhu jednotlivých procesů. Důležitá je dobrá znalost všech implementovaných procesů včetně potřebných vstupů a očekávaných výstupů. Jednotlivé procesy a jejich vzájemnou vazbu lze graficky znázornit prostřednictvím procesního modelu. Model životního cyklu vývoje softwaru znázorňuje procesy, které se zabývají tvorbou softwarového produktu od jeho prvotního záměru až do uvedení na trh. Nabízí plánovitý, metodický postup pro definici pracovních kroků. Tato kapitola popisuje vybrané existující vývojové modely.

## 1.1 *Model velkého třesku*

Tento model je charakterizován neuspořádaným vývojem. Je zde absence pečlivě promýšlených posloupností v organizaci práce. Princip je založen na seskupení lidí, kteří mají realizovat převratný nápad. Společnost jim poskytne finanční prostředky a očekává, že bude vytvořen dokonalý softwarový produkt.

Metoda velkého třesku se vyznačuje především jednoduchostí. Plánování a rozvrhování práce je užíváno minimálně. Veškeré úsilí je zaměřeno na konečný výsledek realizace nápadu. Někdy je výchozí software opravdu kvalitní. Ve většině případů ovšem obsahuje velký počet chyb. To je způsobené tím, že není prováděno žádné nebo téměř žádné testování.

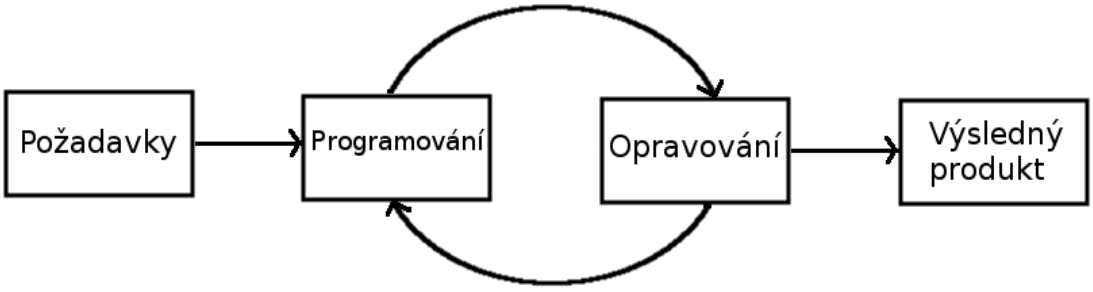
Model je vhodné vybrat pro malé projekty, u kterých není uvedeno konečné datum odevzdání. [1]


*Obr. 1.1: Model velkého třesku*

## 1.2 *Model programuj a opravuj*

Model programuj a opravuj je o něco málo propracovanější než model předchozí. Již se zde vyžaduje hrubý návrh na specifikace produktu. Po stanovení všech požadavků následuje opakující se cyklus složený z programování a opravování vzniklých chyb. Když nastane termín odevzdání produktu zákazníkovi, je výrobek uvolněn na trh a to i za předpokladu, že stále může obsahovat chyby. Aktualizováním softwaru se tyto chyby odstraní. Ani v tomto modelu není zakomponováno důkladné testování, ale tvoří pouze mezičlánek mezi programováním a opravováním.

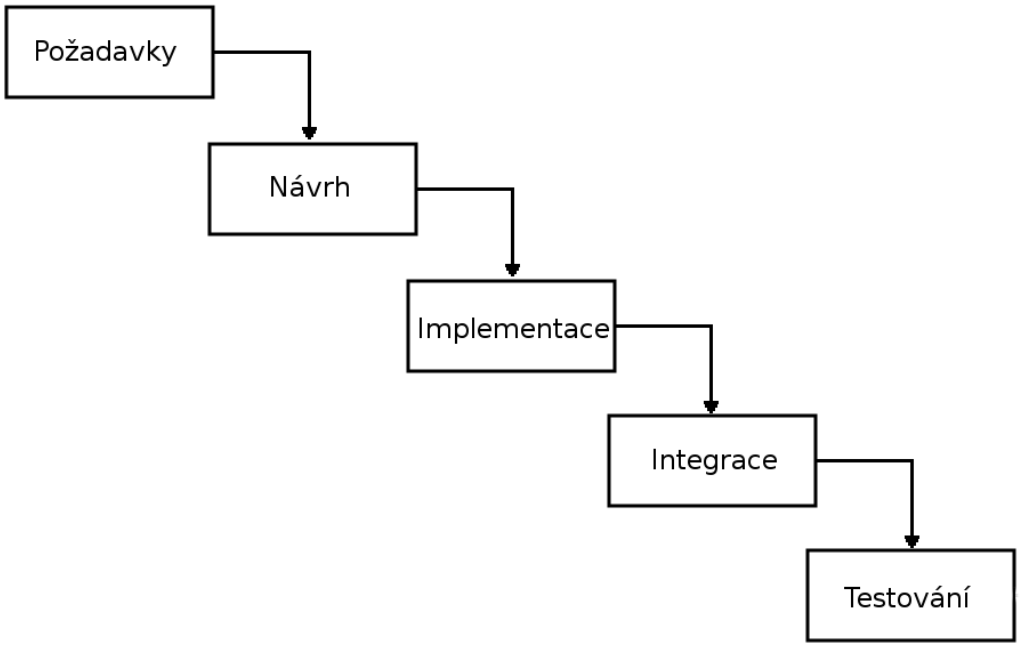
Model se může použít pro malé projekty, které mají být rychle zhotoveny a nevadí, že se chyby odstraní později. Typickým příkladem je vytvoření demonstračního programu. [1]




*Obr. 1.2: Model programuj a opravuj*

## 1.3 *Model vodopádu*

Vodopádový model je nejjednodušší systematický model pro popis procesů vývoje. Vznikl v roce 1970 a jeho tvůrcem byl Winston W. Royce. Jeho název vychází z přirovnání posloupnosti jednotlivých kroků k protékající vodě vodopádem. Všechny fáze, viz obr. 1.3, mají pevně stanovené pořadí a v konkrétním projektu se v určitý okamžik pracuje jen v jedné fázi. V každé etapě vývoje je vymezeno, jaké cíle by měli být splněny a jaké vstupy jsou potřebné k práci. Když je etapa dokončena, je kompletně připravená a zkontrolovaná, může se přikročit k fázi následující. Po opuštění fáze se do ní nelze vrátit a je dovoleno pracovat pouze v aktuální části vývoje. [1, 2]




*Obr. 1.3: Model vodopádu*

Fáze testování se ve srovnání s výše popsanými modely objevuje nově a ve své době to bylo považováno za pokrok, protože se tím redukoval výskyt chyb. Zůstává tam ale problém s nešťastným umístěním na konci vývojového modelu. Testovat se začíná až ve chvíli, kdy je výrobek téměř hotov a připraven na předání klientovi. Opravy chyb v této etapě jsou nejenom finančně ale i časově náročnější, než ve fázi, ve které chyba vznikla. Čím dříve se chyba odhalí, tím méně se bude muset vynaložit finančních prostředků na její nápravu. V každé fázi vývoje se náklady na opravu chyb zvýší až na desetinásobek. Pokud by odstranění chyby ve stádiu sestavování specifikace stálo 5 Kč, náprava totožné chyby objevené ve fázi návrhu by stála

50 Kč. Nejvyšší částka by se musela zaplatit, pokud by tu samou chybu odhalil zákazník. Testování předchází snižování zisku společnosti. [1]

Náklady jsou spojeny i s jakoukoliv změnou ve specifikaci požadavků. Jakmile jsou sestaveny a schváleny, je tím uzavřena první fáze vývoje a požadavky už nelze změnit. Model se tím tak stává značně nepružným. Pokud zákazník přijde kdykoliv během realizace projektu se změnou v zadání, musí se znovu projít všemi etapami vývoje. Opět se musí vytvořit příslušné dokumenty a musí proběhnout schvalovací procedura. Pokud dojde k nepochopení klientova zadání, vyjde tato skutečnost najevo často až při předání. Toto riziko může zapříčinit neúspěch celého projektu a vyústí v jeho kompletní přepracování. Následky jsou pak velmi negativní. Dochází jednak k finančním ztrátám společnosti, může být snížena její důvěryhodnost a termín dodání se musí posunout.

Model je vhodný aplikovat u rozsahově malých a specifikačně jednoduchých projektů, kde budou pevně stanoveny požadavky na produkt, u kterých bude jistota, že se až do konce vývoje nezmění.

Vodopádový model byl postupem času modifikován a existují verze, kde se vyskytuje možnost vrátit se v procesním modelu o jeden krok zpět. Tato výhoda umožňuje provést potřebné úpravy či odstranit případné chyby. [2]

## 1.4 *Spirálový model*

Spirálový model zavedl Barry Boehm v roce 1986. [1] Vychází, mimo jiné, i z vodopádového modelu, odkud převzal definování posloupnosti procesů, ať už se jedná o návrh, vývoj, či testování. Model je navržen tak, aby výrazným způsobem řešil slabé stránky svého předchůdce.

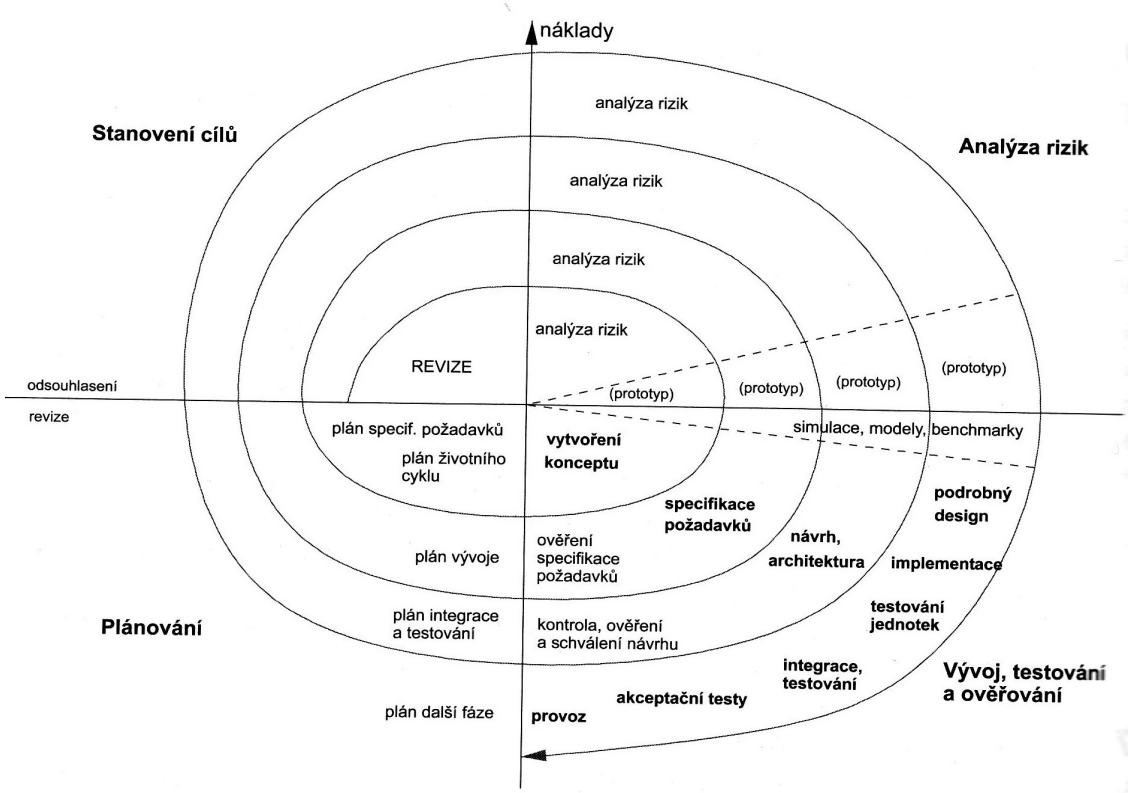
Objevuje se zde iterace, analýza rizik a prototypování. Iterativní vývoj je založen na opakování čtyř fází ve spirále, konkrétně se jedná o určení cílů, rozbor rizik, vývoj a ověřování, plánování. Při každém opakování těchto procesů je docíleno, že se projekt může zpracovávat na vyšším stupni zpřesnění a zdokonalení. Na začátku každé iterace jsou stanoveny nejenom cíle, ale i alternativy, které nabízejí více způsobů řešení, a omezující podmínky pro danou etapu vývoje. Fáze ověřování poskytuje zákazníkovi zhodnocení výstupů projektu, ještě před tím, než vstoupí na další úroveň vývoje.

Analýza rizik má za úkol identifikovat všechna potenciální ohrožení projektu. Důsledkem je včasné vyloučení nevhodného postupu. Riziko představuje jakoukoliv událost nebo situaci, která může zapříčinit nesplnění stanovených cílů. U každého rozpoznaného rizika se určuje jeho nebezpečnost a pravděpodobnost výskytu. Rozeznává se několik typů rizik:

- projektová rizika – snížení rozpočtu, odchod lidí z vývojového týmu atd.,
- technická rizika – selhání hardwaru, nevyzkoušené technologie, problémy se softwarem atd.,
- obchodní rizika – uvedeni na trh konkurenčního produktu, špatné odhady odbytu atd.

Na základě pravidelné a důsledně prováděné analýzy všech rizik se rozhoduje o budoucím směru vývoje, změně podmínek a zdrojů, v krajním případě dokonce i o zastavení projektu. Spirálový model díky tomu umožňuje pružně reagovat na změny a přizpůsobovat projekt aktuálním potřebám.

Na obr. 1.4 je zobrazena struktura spirálového modelu. Radiální rozměr znázorňuje časové i finanční náklady na realizaci projektu.




*Obr. 1.4: Spirálový model (převzato z [3])*

Produkt je pravidelně testován už od počátečních fází vývoje. Tím je zajištěno včasné nalezení chyb a snížení nákladů na jejich odstranění.

Mezi nevýhody modelu patří jeho složitost, která se projevuje především u malých projektů. Administrativa zpomaluje vývoj a snižuje efektivitu práce. Další handicap je závislost na rizikové expertize. Chybně vypracovaná riziková analýza a pozdní odhalení rizika s vysokým stupněm nebezpečnosti může mít negativní vliv na řešení celého projektu.

Vzhledem k tomu, že model klade důraz na plánování, ověřování a analýzu rizik, je vhodné ho použít u rozsáhlých a složitých projektů. Dále se může aplikovat u projektů, kde se předpokládají časté změny ve specifikaci požadavků a častá komunikace se zákazníkem. [1,2,3]

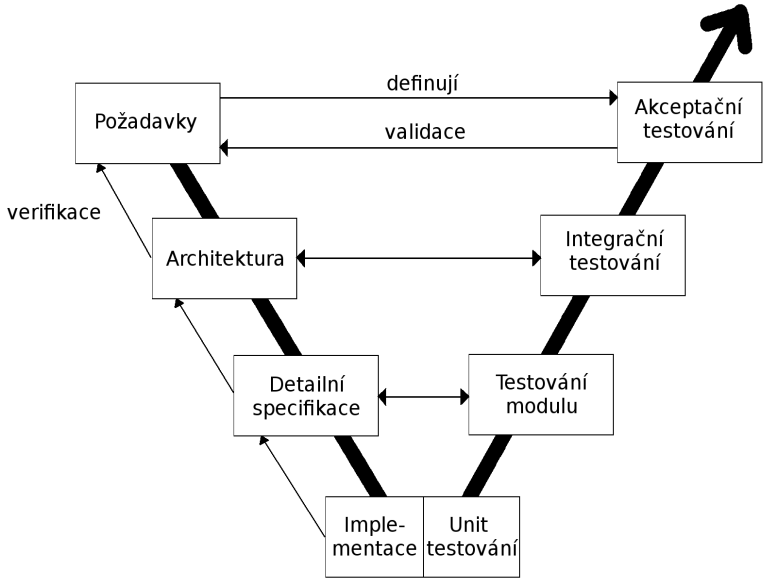
## 1.5 *V-model*

V-model vyvinula německá společnost IABG v roce 1986. [20] Nechala se inspirovat vodopádovým modelem a jeho koncepci rozšířila a vylepšila. Místo lineárního postupu od shora dolů je model zalomen a vytváří tím písmeno „V“, podle kterého je i pojmenován. Písmeno „V“ také představuje symbol pro opakovaně prováděné činnosti validace a verifikace. Vzhledem k tomu, že struktura V-modelu je založena na důkladném prověřování výstupů jednotlivých procesů, využívá se hlavně v oblastech, kde je kladen důraz na kvalitu z hlediska testování.

Jednotlivé procesy jsou vzájemně provázány a tvoří dvě větve, viz obr. 1.5. Levá větev zobrazuje fáze vývoje softwaru a pravá větev k nim přiřazuje fáze testování. Po dokončení každé fáze vývoje softwaru následuje její verifikace, která ověřuje, zda nevznikly chyby. Při verifikaci se kontroluje, jestli se výstupy z daného procesu shodují s očekávanými vstupy následujícího procesu. Každý proces se musí zpracovávat ve stanovené posloupnosti a nelze ho vykonávat současně s jiným. Vývoj produktu končí ve spodní části V-modelu, kde se zároveň nachází i první fáze jeho testování. V pravé části modelu dochází k testování, které je navrženo podle testovacích kritérií z příslušné úrovně procesu na levé straně. Akceptační testy jsou nadefinovány z celkových požadavků, integrační testy ověřují rozhraní architektury a testy modulů jsou navrženy na základě detailní specifikace. [21, 22] Ověření, zda produkt splňuje všechny požadavky zákazníka, se získává při validaci. Cílem validace je potvrdit, že je produkt správně vytvořený. Při verifikaci se ověřuje, zda se produkt správně vytváří. [5]

Názvy jednotlivých fází a jejich počet se může v literatuře lišit, ale vždy je respektováno

stejné množství kroků na pravé i levé straně. Samozřejmě jsou pokaždé zachovány i typy vazeb mezi procesy.




*Obr. 1.5: V-model*

V-model se od vodopádového modelu liší především v grafickém znázornění a nutnosti zachování stejného počtu procesů vývoje softwaru s procesy testování. Vazby mezi procesy ve V-modelu lze totiž aplikovat i do modifikovaného vodopádového modelu.

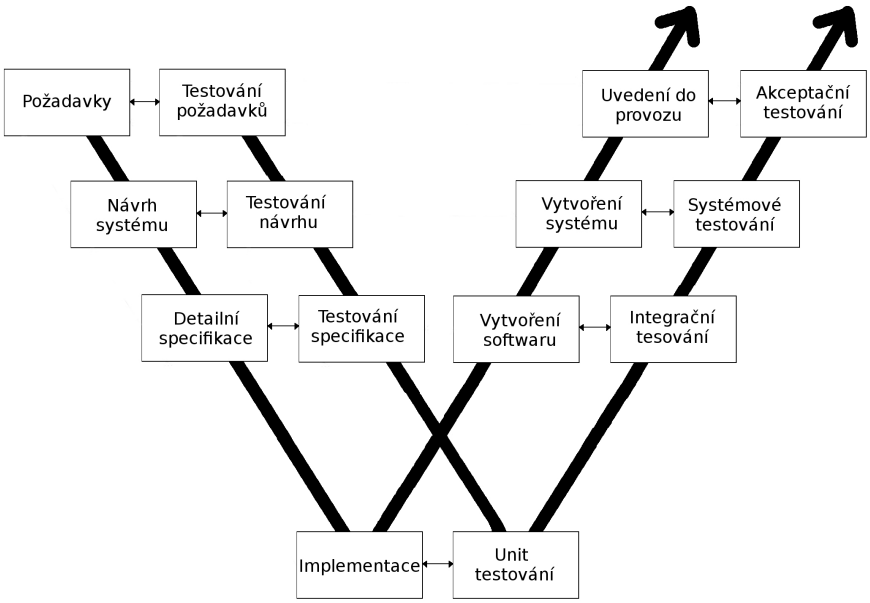
Za nevýhodu u V-modelu je považována, tak jako u vodopádového modelu, nepružná reakce na změny. Pokud dojde k nějaké změně během vyhotovování projektu, musí se upravit všechny příslušné dokumenty včetně plánů testování a musí se znovu vykonat procesy dle posloupnosti od úrovně, kde se změna projevila. Dále model jasně neukazuje, jak odstranit nalezenou chybu. [21, 22] V ZF V-modelu je tato problematika řešena v podpůrných procesech.

Model se doporučuje používat u malých a středně velkých projektů, kde jsou pevné a jasně definované požadavky. [23]

## 1.6 *W-model*

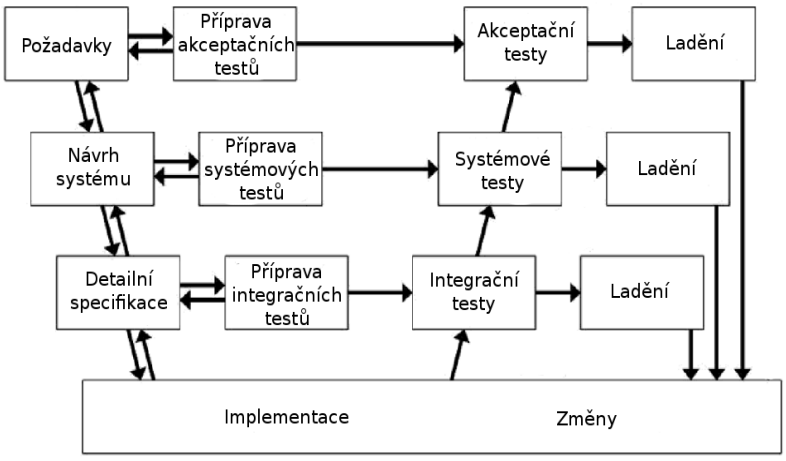
W-model vznikl v roce 1993 a jeho tvůrcem byl Paul Herzlich. Název je odvozen od dvou paralelních „V“, které společně tvoří písmeno „W“. Tento model vychází z koncepce V-modelu a odstraňuje jeho nedostatek v podobě pozdního testování, které přichází na řadu až ve fázi implementace. Ve W-modelu je testování zahájeno hned na začátku projektu. Ke každému vývojovému procesu je přiřazeno příslušné testování, viz obr. 1.6. Účelem je zajistit

splnění stanovených cílů. Na levé straně modelu se nachází statické testování, na straně pravé dynamické tetování. Při statickém testování není vyžadován běh softwaru. Kontroluje se projektová dokumentace nebo samotný kód z hlediska syntaktické správnosti. Dynamické testování se vyznačuje spuštěním softwaru a hledání chyb v jeho funkčním chování. [6]




*Obr. 1.6: W-model*

Další variantu W-modelu představil Andreas Spillner. Nově vytvořil vazbu mezi dynamickým testováním a laděním chyb. Pokud se nalezne během testování chyba, analyzuje se, proč k ní došlo a pak následuje její odstranění. Produkt se vrací zpět do etapy implementace a musí znovu projít všemi testovacími procesy od zdola nahoru. Testeři ověřují, jestli byla chyba napravena a neobjevily se nové problémy. Spillnerův W-model klade důraz na komunikaci mezi různými fázemi vývoje, což je znázorněno na obr. 1.7. [21]




*Obr. 1.7: Spillnerův W-model*

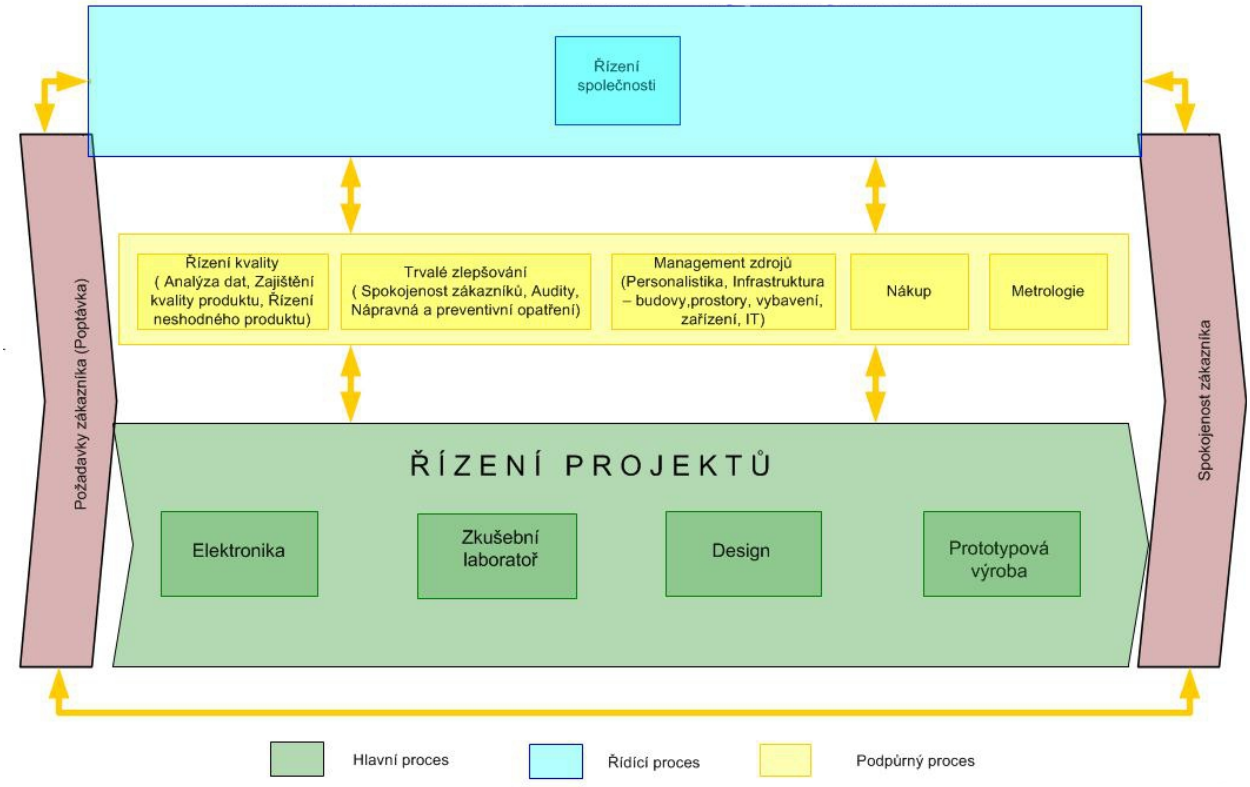
# 2 Popis procesů ve společnosti ZF Engineering Plzeň, s.r.o.

ZF Engineering Plzeň, s.r.o. se zabývá automobilovými řídicími systémy. Nabízí jejich vývoj od návrhu až po výrobu prototypů a testování. Je součástí ZF Group, která sdružuje celkem 121 poboček ve 26 zemích. ZF Group se specializuje na technologie v oblasti podvozků a hnacích soustav a patří mezi top 10 světových dodavatelů pro automobilový průmysl. V roce 2012 bylo dosaženo tržby ve výši 17,4 miliard EUR s přibližně 75 000 zaměstnanci. [26]

V roce 2007 koncern ZF zakoupil v Plzni firmu Value Engineering Services s.r.o, která byla založena dne 6. března 2002. Obchodní název byl změněn na ZF Engineering Plzeň, s.r.o. a společnost byla začleněna do struktury ZF Group. Nyní je ZF Engineering Plzeň, s.r.o. jedním z osmi hlavních vývojových center v rámci ZF Group a orientuje se především na zakázky od své mateřské společnosti ZF Friedrichshafen AG. [29]

V roce 2012 společnost ZF Engineering Plzeň, s.r.o. (dále jen „ZF PLZ“) dosáhla tržby ve výši 169,9 milionu Kč a zaměstnávala zhruba 150 lidí. Firma vlastní certifikát řízení jakosti ISO 9001:2008. [27] Dále plní požadavky norem ISO 170 25:2005 (laboratoř v oddělení mechatroniky) a Automotive SPICE (vývoj softwaru v oddělení elektroniky).

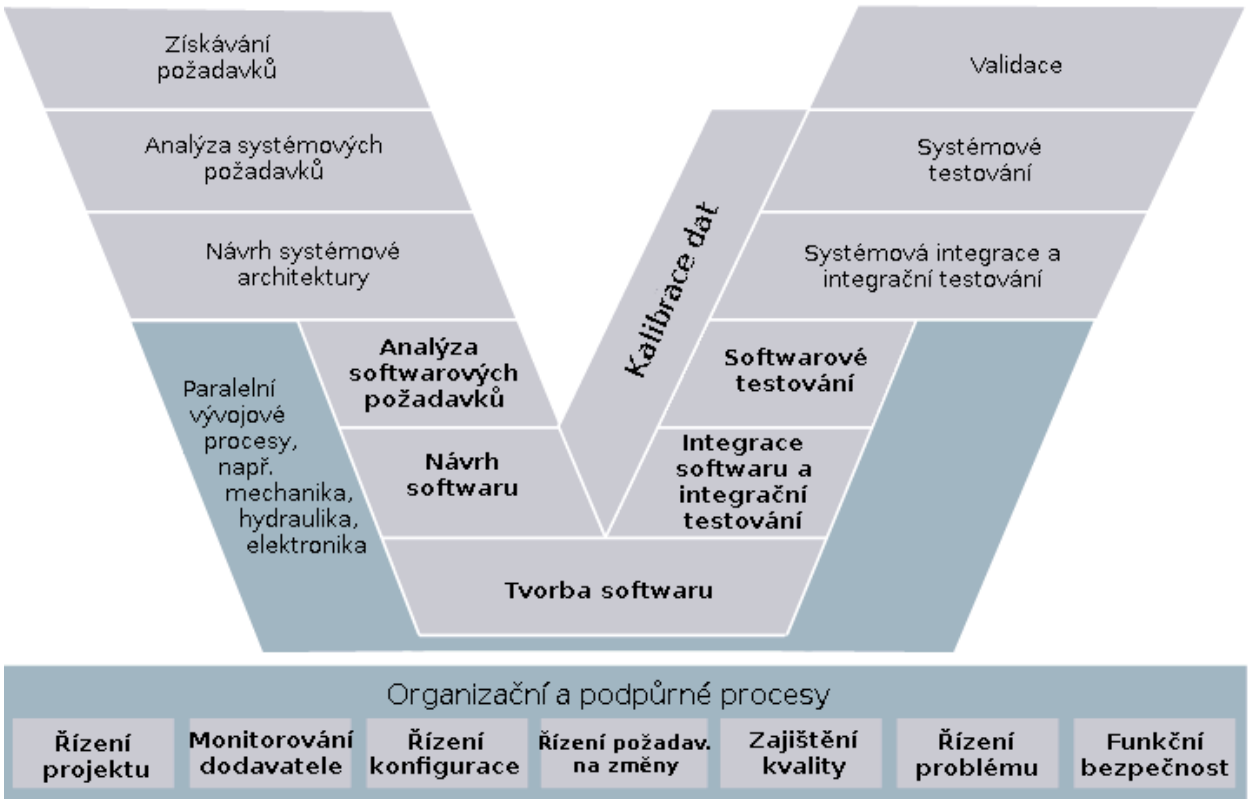
Struktura procesů ve společnosti dle ISO 9001:2008 je zobrazena na obr. 2.1. Procesy jsou rozvrženy tak, aby byly naplněny požadavky zákazníka a bylo docíleno jeho spokojenosti. Existují tři typy procesů – hlavní, podpůrné a řídicí. Hlavní procesy jsou přímo orientované na potřeby klienta a realizují projekty dle jeho zadání. Jsou označovány také jako procesy s přidanou hodnotou, protože jako jediné produkují společnosti zisk. Správné fungování hlavních procesů je zajištěno podpůrnými procesy, které mají na starosti např. dostatek lidských zdrojů a vybavení, kvalitu produktu. Třetí typ procesu je nazván jako řídicí, který je provázán se všemi ostatními procesy a vyznačuje se činnostmi nutnými pro chod celé společnosti. [28] Identifikování všech těchto procesů slouží k lepšímu pochopení fungování společnosti, usnadnění jejich hodnocení a umožnění jejich zlepšování.




*Obr. 2.1: Struktura procesů ve společnosti ZF Engineering Plzeň, s.r.o. (převzato z [29] )*

ZF Friedrichshafen AG poskytuje odborné vedení a předávání dlouholetých zkušeností společnosti ZF PLZ. Do Plzně byl převzat z Německa tzv. standardní proces, který se stal hlavní páteří pro obecný popis všech prováděných procesů při vývoji softwaru v oddělení elektroniky. Standardní proces zahrnuje detailní popsání jednotlivých etap, podle kterých se postupuje při zpracování projektu. Standardní proces byl sestaven na základě mnohaletých zkušeností a odborné praxe německých kolegů a byl přizpůsoben pro plzeňské pracoviště. Pro jeho zobrazení byl zvolen V-model, jehož schéma je znázorněno na obr. 2.2. Každý konkrétní proces má svého vlastníka. Jedná se o osobu, která proces vytváří a zlepšuje za pomoci ostatních pracovníků. [29]

Všechny procesy jsou implementovány v souladu s požadavky Automotive SPICE iniciovanými největšími evropskými výrobci aut. V evropském automobilovém průmyslu se jedná o nejvýznamnější standard řízení procesů a kvality vývoje softwaru. Výsledky hodnocení procesů prováděné podle Automotive SPICE mohou pomoci automobilovým společnostem při výběru svých dodavatelů. Hodnocení dle jednotné metody totiž umožňuje objektivní srovnání mezi jednotlivými dodavateli. [13]




*Obr. 2.2: V-model ve společnosti ZF Engineering Plzeň, s.r.o.*

V-model znázorňuje všechny vývojové a testovací fáze při vytváření celého systému, např. převodovky. Nejprve jsou shromážděny celkové požadavky (od zákazníka, interní, zákonné, z norem) a z nich pak dodavatel vychází při sestavování systémových požadavků. Po navržení a schválení architektury následuje rozdělení systému na podsystémy mechanika, hydraulika atd. Podsystémy se vyvíjejí paralelně a dohromady se slučují při systémové integraci, kde se testují jejich vzájemné vazby. Společnost ZF PLZ se momentálně zaměřuje na podsystém vývoje softwaru. Pro kompletní pochopení procesů ve V-modelu budou v následujících kapitolách popsány všechny jednotlivé fáze, včetně těch, které se v Plzni zatím nevykonávají.

## 2.1 *Získávání požadavků*

První proces ve V-modelu se nazývá „Získávání požadavků“. Pojem požadavek lze vyložit různými způsoby. Dle B. Lawrence je požadavek cokoliv, co ovlivňuje rozhodování při návrhu. I. Sommerville vnímá požadavky jako popis všeho, co by se mělo implementovat. Charakterizují podle něho žádané chování systému a také jeho vlastnosti. Někdy mohou dokonce představovat nějaká omezení procesu vývoje systému. [7] Podle standardu IEEE 610.12-1990 je požadavek definován jako:

- podmínka nebo funkce, kterou potřebuje uživatel k řešení problému nebo dosažení cíle,
- podmínka nebo funkce, kterou musí systém nebo jeho část splňovat, aby vyhověl smlouvě, normě, specifikaci nebo jinému dokumentu, který se na něj formálně vztahuje,
- dokumentovaná podoba některého z výše uvedených dvou bodů. [15]

Uživatelem je myšlen každý účastník projektu, nikoliv koncová skupina lidí, pro kterou je produkt určen.

Požadavky se rozdělují podle různých hledisek do několika kategorií. Základní členění požadavků je na funkční a nefunkční.

- Funkční požadavky popisují funkci a chování, které musí finální produkt za daných podmínek vykonávat.
- Nefunkční požadavky definují vlastnosti celého produktu, konkrétně se může jednat o spolehlivost či bezpečnost. Zároveň obsahují omezující podmínky pro daný systém, např. dostupný hardware pro vývoj a provoz, standardy kvality.

Další možné rozdělení požadavků je podle úrovně na podnikatelské, uživatelské a systémové.

- Podnikatelské požadavky formulují ekonomické, tržní nebo jiné podnikatelské cíle, kterých má být dosaženo.
- Uživatelské požadavky popisují, co bude moci uživatel se systémem provádět.
- Systémové požadavky jsou celkové požadavky kladené na konečný produkt složený i z více podsystémů. Odvozují se z uživatelských požadavků.

Práci s požadavky lze rozlišit na vývoj a jejich následnou správu. Do vývoje požadavků patří činnosti spojené se získáváním, vyhodnocováním, dokumentací a kontrolou požadavků. Cílem této fáze je dosáhnout průkazné shody zákazníka a dodavatele na podobě systému, který se bude implementovat. Typickým výstupem vývoje požadavků jsou dokumenty obsahující např. rozsah a vizi projektu, specifikaci požadavků. Po schválení dokumentů a odsouhlasení požadavků končí jejich vývoj a začíná jejich správa. Do správy požadavků spadají aktivity udržující aktuálnost, integritu a přesnost požadavků. Konkrétně se může jednat o verzování požadavků a celkové dokumentace, řízení změn jednotlivých požadavků, sledování stavu konkrétních požadavků atd. [7]

Cílem procesu získávání požadavků je shromažďovat, zpracovávat a sledovat měnící se potřeby a požadavky zákazníků po celou dobu životního vývoje produktu. Mezi metody, které se

k tomu nejčastěji používají, patří rozhovory a analýzy dokumentů od zákazníka. [13]

Existuje několik rizik, která mohou způsobit nedostatky ve zpracování požadavků a ohrožují tak úspěšný průběh projektu. Jedním z nejběžnějších rizik je nedostatečná komunikace se zákazníkem, která může vést v extrémním případě k vytvoření naprosto jiného produktu. Požadavek obsahuje údaj o tom, co má dodavatel vytvořit. Uživatelské parametry stanovuje klient, který by měl mít co nejpřesnější představu o tom, jaké funkce by měl výrobek mít a k čemu by měl být využíván. Způsob, jakým bude záměr realizován, určuje dodavatel. Někdy mohou být zákazníkovy požadavky omezeny ze strany dodavatele, neboť mohou být např. neproveditelné či netestovatelné. Na základě zákazníkovy charakteristiky vnějšího chování výsledného produktu dodavatel vytvoří vnitřní strukturu výrobku. Pokud zákazník dostatečně spolupracuje s dodavatelem při definici požadavků, neměl by být problém společně stanovit jednotlivé požadavky, které přesně a jednoznačně popisují budoucí výrobek. Všichni účastníci projektu by měli sdílet stejnou představu o tom, co se má vytvořit. Musí se dbát na to, aby byly požadavky řádně zdokumentovány. V praxi tomu tak vždy není. Důvodem může být nedostatek času, nebo neochota zákazníka odsouhlasit všechny detaily na začátku projektu. [7]

## 2.2 *Analýza systémových požadavků*

Proces „Analýza systémových požadavků“ se zaměřuje na transformaci požadavků definovaných zákazníkem na sadu technických požadavků a jejich následnou analýzu z hlediska technické proveditelnosti, rizik a testovatelnosti. [13] Systémové požadavky popisují požadavky pro celkový systém (software, mechanika atd.), proto je na jejich správnou analýzu kladen velký důraz. Termín systém vysvětluje standard IEEE 610.12-1990 jako seskupení prvků organizovaných k dosažení specifické funkce nebo souboru funkcí.

Požadavek by měl být vždy určen přesně a jednoznačně. Jeho formulace nesmí dovolovat více možností výkladů a nesmí obsahovat žádné neurčitosti. Každý, kdo si požadavek přečte, by měl jeho obsah pochopit stejně. Vyjádření typu: „program by mohl být použit pro...“ je nepřípustné. Žádoucí je vyjádřit požadavek jasně, stručně a výstižně. Správně jeho formulace zní: „program bude použit pro...“. Přesné pochopení požadavků umožňuje pracovat na správném problému a najít jeho optimální řešení. Práci pak lze rozvrhnout podle důležitosti a mohou se odhadnout prostředky, které budou během realizace projektu vyžadovány. Pokud nejsou známy všechny požadavky, těžko se odhaduje, kdy a za jakou cenu bude produkt hotový a zda dosáhne

požadovaných cílů. [7]

Mezi důležité vlastnosti požadavků patří:

- konzistentnost – žádný požadavek nesmí být v rozporu s jiným. Možným konfliktem mezi požadavky mohou být různě uvedené vlastnosti výrobku, např. formát výstupní zprávy je v jednom požadavku popsán jako tabulkový a v jiném jako textový. Dalším rozporem může být logický střet mezi dvěma zadanými akcemi, např. jeden z požadavků definuje, aby se vyskytovalo „A“ nebo „B“, zatímco jiný konstatuje, aby se „A“ a „B“ vyskytovaly současně. Konfliktem je i používání odlišných termínů pro popis stejného objektu, např. v požadavku bude zrušení prováděné činnosti nazváno „storno“ a v jiném bude definováno jako „cancel“.
- kompletnost – každý požadavek musí obsahovat úplné informace. Pokud mu chybí nějaké údaje, je označen zkratkou TBD (to be determined). K takovému požadavku pak musí být přiložen popis příčin chybějících informací a následný postup nutný k vyřešení situace a odstranění statusu TBD. Dále je určena zodpovědná osoba za dokončení požadavku ve stanoveném termínu.
- proveditelnost – požadavek musí být možné realizovat v rámci známých možností a omezení projektu.
- dohledatelnost – požadavek musí být označen jedinečným a trvalým identifikátorem, na jehož základě lze vysledovat návrh požadavku, příslušný zdrojový kód, i jeho testování. Identifikátor umožňuje jednoznačné odkazovat na požadavek v průběhu celého projektu. Pokud se nějaký požadavek upraví, lze určit všechny části produktu, kterých se navrhovaná změna dotkne.
- jednoznačnost – každý požadavek by měl mít pouze jednu interpretaci. K jednoznačnosti přispívá používání technického jazyka, který má význam jednotlivých pojmů předem definovaný.
- priorita – požadavky nejsou stejně důležité. Některé jsou pouze žádoucí a jiné mají zásadní význam pro vydání produktu. Každému požadavku by proto měla být přiřazena priorita, aby byly rozdíly snadno rozpoznatelné. Vývojovému týmu to pomáhá činit systematická rozhodnutí o určení pořadí implementace a upřednostňování požadavků v situacích, kdy je např. krácen rozpočet, opožďují se termíny, nebo dochází k přidání

nových požadavků během vývoje.

- nepostradatelnost – požadavek by měl být užitečný, aby zbytečně nekomplikoval vývoj produktu.
- přizpůsobitelnost – struktura požadavku dovoluje snadné doplňování případných změn.
- ověřitelnost – veškeré požadavky by měly být ověřitelné, nejlépe prostřednictvím testů. [7, 16]

Nejednotné, neúplné, neproveditelné a nejednoznačné požadavky ověřit nelze. Příkladem neproveditelného požadavku je tvrzení, že program nikdy nevstoupí do nekonečné smyčky. Testování takové vlastnosti je teoreticky nemožné. Naopak ověřitelným požadavkem je, že se stavová hlášení budou aktualizovat každých 30 sekund po spuštění nějakého procesu na pozadí. Používají se zde konkrétní termíny a měřitelné veličiny. [7]

## 2.3 *Návrh systémové architektury*

Během procesu „Návrh systémové architektury“ se identifikují všechny podsystémy, nadefinuje se mezi nimi rozhraní a přiřadí se jim odpovídající systémové požadavky. [13]

Pojem architektura lze vysvětlit jako základní strukturu systému, danou jeho podsystémy a jejich vazeb mezi sebou a okolím. Cílem vytvoření architektury systému je získat představu o tom, jak se bude daný produkt tvořit. K zachycení těchto informací se obvykle používají obrázky, náčrty a diagramy. Návrh systémové architektury podsystémů musí korespondovat se systémovými požadavky z předchozího procesu. To se zajišťuje pomocí kontroly, tzv. review. Nezávislá osoba s příslušnými kompetencemi přezkoumává pracovní výstupy někoho jiného formou odpovídání na předem připravené otázky. Dále musí být vyvinuta ověřovací kritéria (pokud nejsou na první pohled zřejmá) pro integrační testování systému. Kritéria definují, co se má testovat. O způsobu provedení testů rozhodují testeři. [8, 13, 29]

## 2.4 *Analýza softwarových požadavků*

Prvním prováděným procesem v ZF PLZ je „Analýza softwarových požadavků“. Správně sestavené požadavky jsou základem úspěšného dokončení celého projektu, tzn. produkt se ve stanovený termín za sjednanou cenu dodá zákazníkovi a splní jeho nároky na funkce a kvalitu. Klíčovou roli požadavků v softwarovém projektu popsal F. Brooks: *„Nejtěžší*

*samostatnou fází stavby softwarového systému je rozhodnout, co přesně má vzniknout. Žádná z ostatních částí konceptuální práce není tak složitá, jako vybudování podrobných technických požadavků včetně všech rozhraní k lidem, strojům a dalším softwarovým systémům. Žádná z ostatních částí práce systém tak nezmrzačí, když ji uděláte špatně. Žádná z ostatních částí se tak těžko neopravuje později“.* [7]

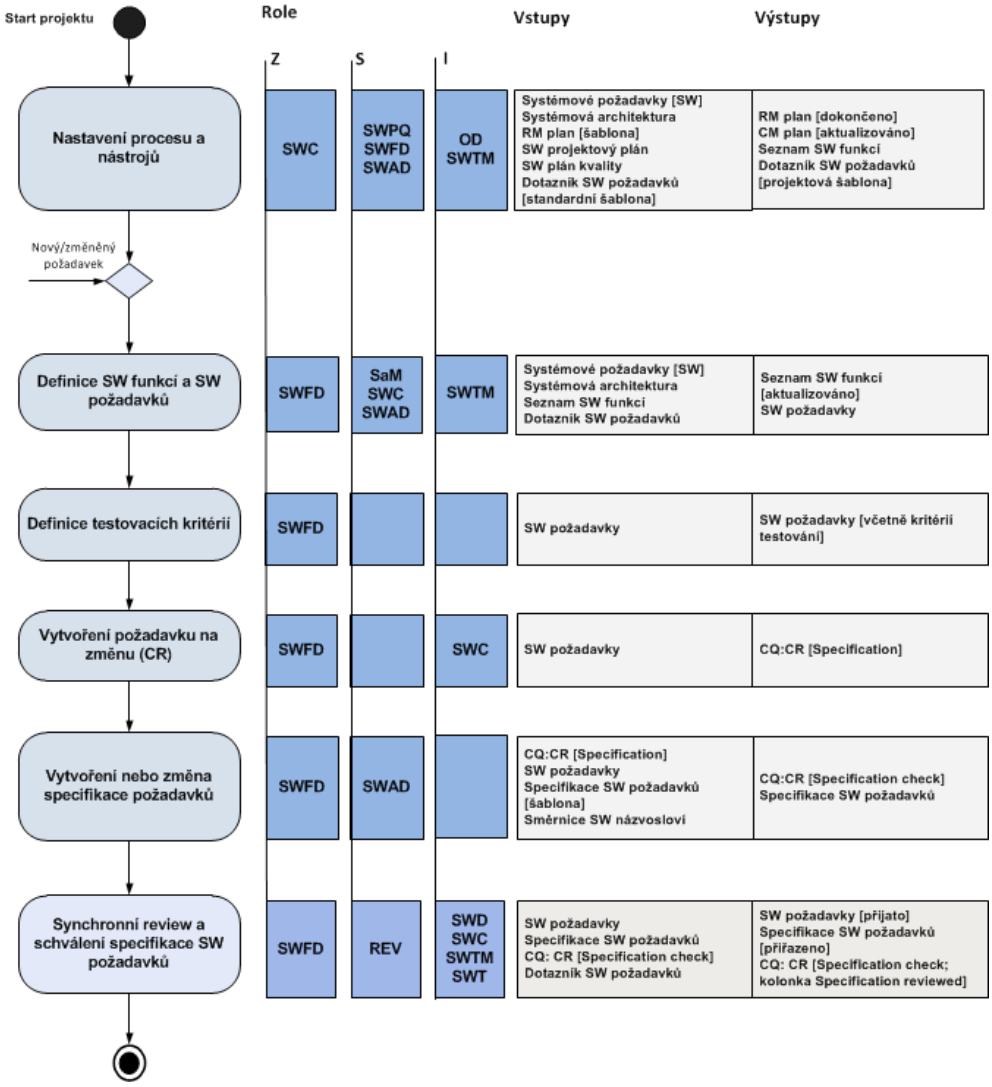
Pokud jsou požadavky kvalitně napsány, plyne z toho řada výhod. Mezi ně lze zařadit snížení objemu předělávané práce, nižší počet chyb v požadavcích, menší množství zbytečných funkcí, rychlejší vývoj produktu, přesnější odhady. Také se tím předchází nedorozuměním, která snižují kvalitu produktu a jeho hodnotu pro zákazníka. Dále se snižuje výskyt chaosu v projektu a výše nákladů na vylepšování požadavků. Definování požadavků bývá časově náročné, musí proběhnout několik diskuzí. To lze zanedbat v porovnání s několikanásobně vyšší časovou náročností potřebnou na opravu chybných požadavků objevených v rámci testování nebo uvolnění výrobku.

Na základě požadavků se vypracuje jejich specifikace (SRS), která může být ve formě dokumentu, databáze či informací uložených v komerčním programu pro správu požadavků. Jedná se o detailní popis očekávaného chování a vlastností finálního produktu, jenž má být vyvinut. SRS zahrnuje všechny softwarové funkční i nefunkční požadavky. Neobsahuje však podrobnosti o postupu testování, plánování projektu, detaily o návrhu či implementaci. Specifikace je hlavní zdroj požadavků, musí být podrobná a pokud v ní nějaké informace scházejí, v žádném případě by neměly být domýšleny zákazníkem ani vývojovým týmem. Pokud se tedy požadovaná funkce neobjeví ve specifikaci, nikdo by neměl očekávat, že bude začleněna do výsledného produktu. [7]

Jak již bylo řečeno na začátku této podkapitoly, prvním prováděným procesem ve společnosti ZF PLZ v rámci vývoje podsystému software dle užívaného V-modelu je „Analýza softwarových požadavků“. Jejím cílem je jasně stanovit softwarové požadavky pro celý systém. Musí být vymezeny funkční a nefunkční softwarové požadavky, které vycházejí ze systémových požadavků a návrhu architektury systému. Mezi těmito požadavky nesmí vzájemně docházet k rozporu a vždy se musí dbát na jejich soulad. Následuje jejich analýza z hlediska technické proveditelnosti, rizik a testovatelnosti. Také se zhodnocuje dopad softwarových požadavků na provozní prostředí a stanoveným požadavkům se přiřadí priorita. Požadavky jsou dodavatelem i zákazníkem schváleny a v případě potřeby se mohou aktualizovat. Mezi softwarovými a

systémovými požadavky a také mezi softwarovými požadavky a architekturou systému je zajištěna bilaterální dohledatelnost, tzn. vzájemné vazby jsou sledovány v obou směrech. Pokud dojde ke změně softwarového požadavku, je analyzován z hlediska ceny, časové náročnosti a technického dopadu na systém. Dále musí být zajištěno, aby byly změny přístupné všem relevantním účastníkům projektu. [13, 29]

Každý vykonávaný proces ve společnosti ZF PLZ má předem jasně stanovený postup. Schéma posloupnosti jednotlivých pracovních kroků při provádění analýzy softwarových požadavků je znázorněn na obr. 2.3. Grafická podoba procesu umožňuje lepší porozumění všech souvislostí mezi jednotlivými činnostmi prováděnými v daném procesu. U každé činnosti je přesně nadefinováno, kdo je zodpovědný za její realizaci, s kým musí dotyčný spolupracovat a kdo musí být informován o jejím dokončení. Dále jsou určeny vstupní a výstupní dokumenty.




*Obr. 2.3: Vývojový diagram procesu „Analýza softwarových požadavků“*

Podle vývojového diagramu musí být u každého projektu nejprve nastaveny prováděné procesy a nástroje, s kterými se bude pracovat. Komunikaci mezi softwarovými požadavky, jejich změnami a jinými relevantními částmi zajišťuje nástroj pro správu požadavků IBM Rational DOORS. Nástroj umožňuje uchovat, sledovat a řídit požadavky. Dále poskytuje možnost komunikace s jinými nástroji, např. IMB Rational ClearQuest. Každý člen projektového týmu musí být proškolen v zacházení s nástrojem DOORS a musí v něm mít zřízena potřebná práva. V rámci nastavování procesů a nástrojů, je vytvořeno několik dokumentů, např. RM plan (Requirement Management Plan), kde je popsán postup pro použití nástroje DOORS v konkrétním projektu a instrukce pro analýzu softwarových požadavků.

V dalším pracovním kroku se nadefinují softwarové funkce, které se musí popsat z hlediska funkcionality, rozhraní a systémových omezení. Ustanoveny jsou i softwarové požadavky.

Následující činnost je zaměřena na určení kritérií softwarového testu, který vymezuje, co se bude testovat, nikoliv jak bude testování probíhat. U funkčních požadavků lze kritéria testování snadno odvodit, proto nemusí být předem definovány a tento pracovní krok se může vynechat. To ovšem neplatí u nefunkčních požadavků, kde musí být předmět testování vždy přesně předem určen.

V dalším pracovním kroku dochází k přidání nového požadavku na změnu softwarové funkce.

Následující činnost se soustředí na vytvoření nebo změnu funkční specifikace. Před implementací se specifikace softwarových požadavků zkontroluje a schválí. Kontroly (review) se v ZF PLZ provádějí nejčastěji dvěma metodami. Obě vedou ke zlepšování kvality dokumentů. Pomáhají odhalit jejich chyby, nejednoznačnost a jiné problémy. První z nich je Author Reader Principle (ARP). Jedná se o systematickou formální metodu, kdy autor předá dokument čtenáři ke kontrole a čeká na jeho zpětnou vazbu. Druhá metoda nazvaná Walkthrough (WT) je systematická ale méně formální. Autor detailně seznámí dotyčného s dokumentem a na základě diskuze se odstraní případné nedostatky. Podle toho, zda je review plánované či nikoliv, se rozlišuje na asynchronní a synchronní. [29]

## 2.5 *Návrh softwaru*

Proces „Návrh softwaru“ vychází ze specifikace softwarových požadavků a orientuje se na sestavení architektury softwaru. Pojem softwarová architektura lze vyložit jako základní

uspořádání podsystému, dané jeho komponentami, vzájemnými vztahy těchto komponent a jejich vztahy k okolnímu prostředí a zásadami řídící její návrh a vývoj. Komponenta je softwarový element, který může být opětovně použit v jiném podobném návrhu nebo přizpůsoben za účelem vytvoření nových vlastností a funkcí. Dle L. Basse je softwarová architektura dána strukturou, zahrnující softwarové prvky, jejich vnější vlastnosti a vazby mezi prvky. [8] Z obou vysvětlení vyplývá, že se architektura zabývá nejen strukturou, ale i chováním jednotlivých částí celku.

Existuje několik aspektů, které by se při návrhu softwaru měly zvážit, aby bylo dosaženo očekávaných výsledků. Mezi ně patří:

- kompatibilita - software je schopen spolupracovat s dalšími produkty, např. část softwaru může být kompatibilní s její starší verzí,
- rozšiřitelnost - možnost přidávat nové funkce bez větších změn základní architektury,
- odolnost proti poruchám - software je proti poruchám odolný a pokud k nim dojde, je schopen obnovy,
- spolehlivost - software je schopen plnit požadovanou funkcionalitu za stanovených podmínek,
- opakovatelná použitelnost – v případě potřeby mohou být komponenty znovu použity. [24]

Přínosem návrhu softwaru je možnost určit dopady případných změn ještě před tím, než skutečně nastanou. Architektura identifikuje komponenty, jejich interakce, závislosti a současně umožňuje zpětné vysledování odpovídajících požadavků. Na základě těchto informací lze analyzovat změnu požadavku z hlediska dopadu na komponenty, které se podílejí na realizaci daného požadavku. Analýza může významně pomoci s odhalováním rizik spojených s provedením určité změny, se stanovením vlivu této změny na systém a s výpočty odhadu nákladů na její zavedení.

K návrhu softwaru se používá komplexní modelovací a vývojářský nástroj Enterprise Architect od společnosti Sparx Systems, který podporuje tvorbu vývojových diagramů a potřebných schémat v jazyce UML.

Struktura softwarové architektury je sestavena na základě funkčních a nefunkčních

požadavků a je složena ze softwarových komponent. Jednotlivým komponentám jsou přiřazeny příslušné požadavky, což zvyšuje míru přehlednosti, dohledatelnosti a usnadňuje to práci vývojovému týmu. Interakce mezi komponentami jsou nadefinované skrz rozhraní. Vzájemné působení dynamického chování komponent je zhodnoceno a zdokumentováno. Dynamické chování je určeno provozními režimy, např. spuštění, vypnutí, kalibrace. Musí být jasně nadefinovány cíle spotřeby zdrojů pro softwarové komponenty, např. ROM, RAM, CPU. Dále je zpracován detailní design jednotlivých softwarových komponent. Navrhují se i kritéria pro verifikaci dynamického chování každé komponenty. Celý proces je zakončen schválením softwarového designu a ověřením, zda byly splněny všechny softwarové požadavky.

Během návrhu se specifikují a vypočítávají technicko-ekonomické parametry, nalézají se nejvhodnější řešení jednotlivých funkcí, vybírají se použité technologie a vypracovává se technická dokumentace. [8, 13, 29]

## 2.6 *Tvorba softwaru*

Proces „Tvorba softwaru“ se zaměřuje na samotné programování, tj. vytváření zdrojového kódu v určitém programovacím jazyce. Ve společnosti ZF PLZ jsou aplikace psány především v jazyce C.

Proces zároveň zahrnuje i první testování naprogramovaných funkcí. V souvislosti s testováním je nutné objasnit několik základních pojmů. To, že se testování spojuje s odhalováním chyb, je zřejmé, někdy se ale v definicích liší rozsah a způsob v jejich vyhledávání. Testování může být popsáno jako dynamická kontrola očekávaného chování aplikace podle specifikace. Program lze tedy otestovat pouze po jeho spuštění a musí existovat soubor podrobně zpracovaných požadavků, oproti kterým bude aplikace kontrolována. Více obecný pohled na testování nabízí B. Hetzel, který testování definuje jako veškerou aktivitu zaměřenou na vyhodnocení vlastností programu nebo systému a určení, jestli odpovídají očekávaným výsledkům. [19] Další širší náhled na testování poskytuje B. Zelinka, který tvrdí, že testování je *„úsilí vyvinuté ke zjištění toho, zda systém pracuje tak, jak je popsáno v jeho návrhu“*. [19] V obou definicích je kromě dynamického testování zahrnuto i statické, které k vyhledávání chyb nevyžaduje spuštění kódu.

Dále je nutné definovat pojem chyba v softwarovém inženýrství. O chybu se jedná,

pokud je splněna alespoň jedna z uvedených podmínek:

1. Podle specifikace nedělá software něco, co by měl.
2. Software se chová tak, jak by se podle specifikace chovat neměl.
3. Softwarová aplikace má funkce, které nejsou ve specifikaci uvedeny.
4. Specifikace je neúplná.

Použití jednotlivých podmínek může být demonstrováno na jednoduchém příkladě. Ve specifikaci produktu je uvedeno, že se aplikace nezablokuje a spočítá objem krychle. Pokud je výpočtem nesprávný údaj, jedná se o chybu podle prvního pravidla. Po zadání čtyřmístného čísla určující délku hrany krychle se program přetíží a zablokuje. Toto selhání patří do skupiny chyb podle druhého pravidla. Pokud program umožňuje vypočítat i povrch krychle, nastane chyba podle třetího pravidla. Dlouhé čekání na zobrazení výsledku objemu může být chybou identifikovanou podle čtvrtého pravidla. Doba odezvy výpočtu měla být specifikována, ale byla opomenuta. [1]

Při testování softwaru se objevují pojmy černá skříňka (tzv. black box) a bílá skříňka (tzv. white box). U testování černé skřínky se analyzují vstupy a výstupy bez znalosti vnitřního algoritmu. Produkt je černou skříňkou, do které se nelze podívat a určit, jak uvnitř pracuje. Lze pouze pozorovat, jaký se získá výsledek po vložení vstupních dat. Naopak u testování metodou bílé skříňky je zdrojový kód k dispozici. Je umožněno vidět dovnitř skříňky, poznat vnitřní strukturu softwaru, nahlížet na použité algoritmy a lépe tak vyhledávat chyby v aplikaci. Kombinace dvou předchozích případů se nazývá testování šedé skříňky (tzv. grey box). Kontrole jsou podrobovány vstupy a výstupy s částečnou znalostí struktury zdrojového kódu. [1]

K testování různých úrovní vývoje softwaru je využíváno několik typů simulace:

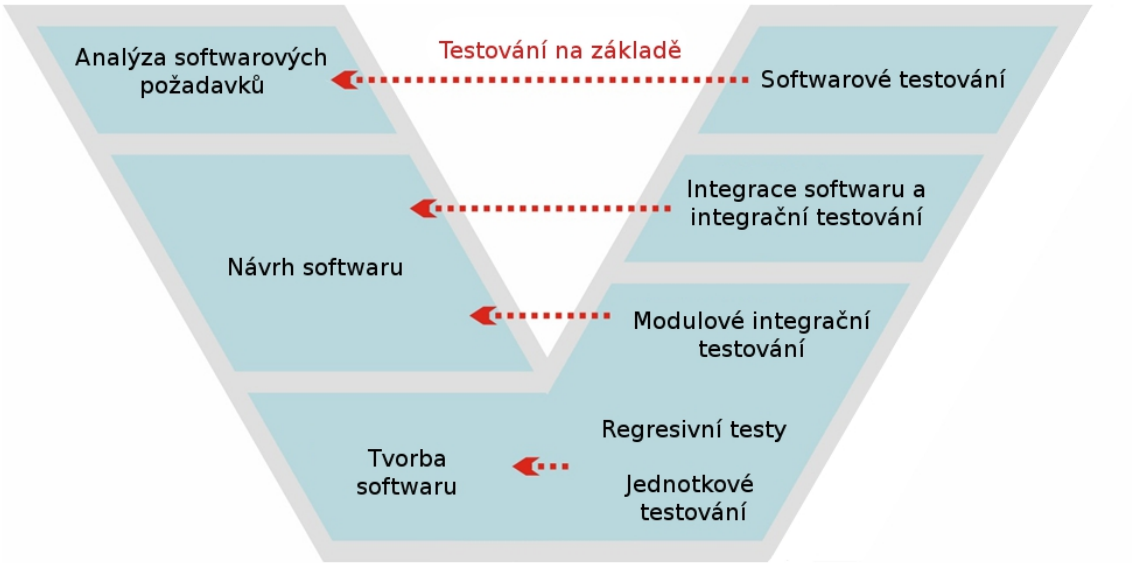
- Model in the loop (MiL) – v tomto případě je sestaven model řídících algoritmů (např. v nástroji Matlab SIMULINK) a je spuštěna jeho simulace, která ověřuje, zda jsou splněny požadavky. Návrh modelu je testován prostřednictvím testovacích případů (test case), kdy jsou zadány určité hodnoty proměnných a je sledováno, zda jsou vypočítané výsledky ve shodě s očekávanými. Metoda testování MiL je zaměřena na simulaci chování modelu s cílem dosáhnout požadovaného chování, ale ne nutně v reálném čase.

- Software in the loop (SiL) – jedná se o typ simulace, kdy je testován samotný zdrojový kód, který může být vygenerovaný na základě modelu řídících algoritmů nebo je ručně napsaný.

- Hardware in the loop (HiL) – jedná se o metodu testování konkrétních řídících jednotek, ke kterým je připojeno zařízení (např. motor) simulující určité elektromechanické prostředí. Testování probíhá v reálném čase a v laboratorních podmínkách. Díky tomu lze zjistit, jaké bude chování řídící jednotky ve skutečném automobilu nejenom za běžných podmínek provozu ale i za extrémních stavů. [29, 32]

Během procesu tvorby softwaru nejprve vývojář vytvoří zdrojový kód podle softwarového návrhu a poté jeho správnost prověří tzv. jednotkovými testy (unit testy), viz obr. 2.4. Za jednotku se považuje samostatně testovatelná část zdrojového kódu, např. funkce. Každá funkce by se měla izolovat od ostatních částí programu, aby byla zajištěna její nezávislost. To není snadné, neboť jednotlivé jednotky mezi sebou komunikují, dostávají vstupy z jiných částí programu a jejich výstupy mohou být určeny pro další zpracování jiných jednotek. Problém se řeší pomocí nástrojů, které jsou schopny simulovat požadované vstupy a ty jsou pak jednotlivým jednotkám přiděleny. Po celou dobu unit testů má vývojář přístup k vnitřní struktuře programu, jedná se tedy o testování bílé skříňky. Pokud dojde k nějaké změně softwaru, musí být znovu otestován. Zda byly otestovány všechny části kódu kontroluje test pokrytí (test coverage). [1, 29, 33]

Zdrojový kód se podrobuje kontrole ještě před jeho samotným spuštěním. Mezi konkrétní prováděné testy metodou statické analýzy kódu patří např. polyspace, MISRA-C 1998, kontrola názvů (naming rules check). Polyspace má prokázat ve zdrojovém kódu nepřítomnost chyb za běhu programu (run-time errors), např. dělení nulou. MISRA-C 1998 je standard obsahující 93 povinných a 34 doporučených pravidel pro vývoj softwaru automobilového průmyslu. Pomáhá k zajištění vyšší bezpečnosti a spolehlivosti zdrojového kódu. Naming rules check kontroluje správné označení proměnných. [29, 34]




*Obr. 2.4: Přehled prováděných testů ve V-modelu*

Na úrovni tvorby softwaru se ještě vykonávají další typy testů - regresivní a modulové integrační. Odpovědnost za jejich provedení má obvykle samotný vývojář. Regresivní testy se používají u softwaru, ve kterém došlo ke změně vlastností přidáním či odebráním funkce, nebo úpravou stávajících funkcí. Úkolem těchto testů je ověřit, že zásahy provedené v aplikaci nenarušily funkcionalitu ostatních částí programu, na které zásahy nemají mít vliv. Zjednodušeně řečeno, testuje se, jestli přidání funkce nebo její oprava nezapříčinily novou chybu ve funkční části programu.

Modulové integrační testy se, na rozdíl od dvou předchozích případů, testují proti detailní specifikaci návrhu softwaru. Modul vznikne sloučením předem určených jednotek do jednoho souboru. Tyto testy prověřují interakce mezi jednotlivými částmi kódu, které byly podrobeny unit testování. [29]

## 2.7 *Softwarová integrace a integrační testování*

Další proces je nazván „Softwarová integrace a integrační testování“. Na této úrovni testování mohou být moduly sjednocovány do větších celků, tzv. komponent a jsou otestovány proti specifikaci návrhu celého softwaru. Zda budou testovány komponenty nebo moduly závisí na příslušném projektu. Cílem integračního testování je ověřit, zda mezi sebou komponenty (případně moduly) správně komunikují a v požadovaný okamžik si předem určeným způsobem

předají informace, které mají správný formát i obsah. Testování provádí softwarový integrátor. Pracuje pouze se vstupy a výstupy a nemá k dispozici vnitřní algoritmy, testuje tedy černou skříňku. [13, 29]

Testování je prováděno v souladu s testovací strategií a všechny výsledky jsou dokumentovány. Pokud není výsledek z testů totožný s předpokládaným výsledkem, je analyzováno odpovědnými osobami, zda se jedná o chybu. Po vyhodnocení a potvrzení, že jde o chybu, kterou je nutno odstranit, je vytvořen požadavek na změnu (change request - CR) typu „error“ v nástroji IBM Rational ClearQuest. Tento nástroj umožňuje automatizovanou a sofistikovanou správu změn během kompletního vývoje softwarového produktu. V nástroji lze založit tři typy záznamů:

- CR – umožňuje plánovat, sledovat a kontrolovat požadavky na softwarové změny.
- Job – umožňuje plánovat, sledovat a kontrolovat aktivity softwarového vývoje kromě softwarových změn, např. asynchronní review.
- Release – umožňuje plánovat, sledovat a kontrolovat uvolnění celého softwaru k zákazníkovi.

## 2.8 *Softwarové testování*

Po integračních testech přichází na řadu „Softwarové testování“, které se zaměřuje na aplikaci jako celek v podobě, v jaké bude používán zákazníkem. Ověřuje se, zda je reálné chování softwaru v souladu s chováním očekávaným dle specifikace softwarových požadavků. Testuje se, jestli podsystém plní úlohu, pro kterou byl vyvinut, jestli vrací správná výstupní data a jestli je dostatečně ošetřen proti nestandardním situacím. V případě nalezení chyby se postupuje stejným způsobem jako u předchozího procesu.

Samozřejmostí je zajistit dohledatelnost mezi požadavky a příslušným testováním. Pro každý požadavek je vyvinut alespoň jeden testovací případ, který ověřuje, jestli byl požadavek splněn. Čím více je vytvořeno různých testovacích případů, tím větší je jistota, že byl požadavek správně implementován. [13, 29]

## 2.9 *Systémová integrace a integrační testování*

V procesu „Systémová integrace a integrační testování“ dochází ke spojování všech

podsystémů, např. software, mechanika, hydraulika, elektronika, a testuje se mezi nimi vzájemná spolupráce. Pokud bude některý z podsystémů sám o sobě správně fungovat, ale v rámci celého systému nebude schopen pracovat, je pochopitelně nepoužitelný. V takovém případě se postupuje podle zásad problémového řízení, podle nichž se daná závada odstraňuje. Postup je definován v dokumentu „Problem plan“. [13, 29]

## 2.10 *Systémové testování*

Během procesu „Systémové testování“ je produkt ověřován jako funkční celek z pohledu zákazníka a je posuzováno, zda je připraven pro doručení. Chování celého systému musí odpovídat specifikaci systémových požadavků, kde jsou stanovena kritéria pro realizaci jednotlivých testů. Systémové testy jsou pak navrženy takovým způsobem, aby pokrývaly celou problematiku řešenou v daném projektu. Opět platí, že testování probíhá podle testovacího plánu a všechny výsledky musí být zdokumentované kvůli dohledatelnosti. [13, 29]

## 2.11 *Validace*

Při validaci se provádějí akceptační testy většinou na straně zákazníka a ověřuje se, zda je produkt správně vytvořen před zahájením ostrého provozu a vyhovuje všem požadavkům klienta.

Pokud validace proběhne úspěšně, projekt může být ukončen a produkt je převzat zákazníkem. V případě, že jsou při validaci nalezeny chyby, musí být analyzovány. Zjišťuje se, proč k nim došlo a v jakém podsystému se vyskytují. Při nalezení chyb v této fázi je jejich oprava nejnákladnější. Ke komplexnímu řešení problémů při reklamaci zákazníkem je používána zpravidla metoda 8D-Report. Jejím smyslem je identifikovat, napravit a eliminovat opakování výskytu nedostatků. Zaměřuje se na původ problému určením jeho příčiny a zavádí trvalá nápravná opatření. Struktura 8D-Reportu je rozdělena, jak název napovídá, do osmi částí:

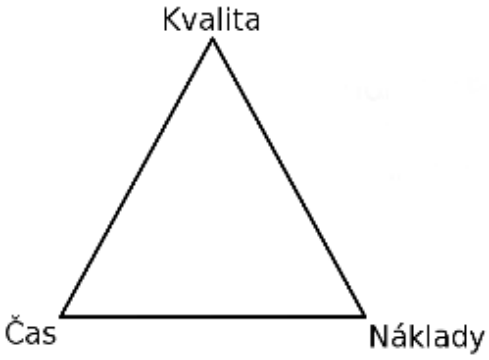
- jmenování členů týmu, kteří budou daný problém řešit,
- popis problému,
- zavedení okamžitého opatření k izolování problému,
- určení kořenových příčin vzniku problému prostřednictvím např. Ishikawova diagramu,

- výběr návrhu opatření, které odstraňuje hlavní příčiny problému z hlediska předpokládané účinnosti a nákladů,
- realizace trvalých nápravných opatření,
- zabránění opětovnému výskytu řešeného problému a potenciálních podobných problémů,
- komunikace se zákazníkem a projektovým týmem, popř. ocenění týmu. [30]

## 2.12 *Řízení projektu*

Řízení projektu lze vyložit podle H. Kerznera jako „*souhrn aktivit, které spočívají v plánování, organizování, řízení a kontrole zdrojů společnosti s poměrně krátkodobým cílem, jenž byl stanoven pro realizaci specifických záměrů.“* [9] Další definice vychází z teorií nejuznávanějšího světového sdružení projektových managerů Project Management Institute (PMI). Řízení projektu představuje „*soubor znalostí, nástrojů, schopností, technologií na činnosti projektu tak, aby byly splněny požadavky projektu.“* [18] Podstata obou definic je stejná. Řízení projektu je soubor aktivit, znalostí a metod, kdy během vynaloženého úsilí jsou materiální i nemateriální zdroje přeměněny na výrobky, služby či jejich kombinace tak, aby k určitému datu byly dosaženy vytyčené cíle. (Pozn. projektové řízení má na rozdíl od řízení projektu širší význam a kromě řízení jednotlivých projektů zahrnuje i jejich koordinaci.)

Hlavními prvky, které ovlivňují hranice projektového prostředí, jsou finanční prostředky, čas a kvalita produktu. Často bývají zobrazovány v podobě trojúhelníku, viz obr. 2.5. Jejich udržování v rovnováze přispívá k úspěšnému ukončení zahájeného projektu. Pro splnění této podmínky slouží plán projektu, podle kterého je sled aktivit a spotřeba zdrojů koordinována a za spoluúčasti kontrolních mechanismů se monitorují. [12]




*Obr. 2.5: Projektový trojúhelník*

V této části práce je nutné objasnit, co znamená pojem projekt. Význam tohoto slova se dříve v projektové praxi ustálil ve smyslu návrh, plán. Toto pojetí vedlo k závěru, že se jedná o dokumentaci, která slouží k posouzení technickoekonomické úrovně návrhu produktu. Termín projekt vychází z anglosaského pojetí slova *project*, které označuje proces plánování a řízení realizace projektu. Nejedná se tedy jen o výsledek v podobě projektové dokumentace, ale o tvůrčí proces. Plánování popisuje žádoucí stav a řízením projektu se tohoto stavu dosahuje. Podle PMI je projekt vymezen jako „*dočasné úsilí vynaložené na vytvoření unikátního produktu, služby nebo určitého výsledku.“* [18] Podobná definice pochází od V. Němce, který projekt vnímá jako „*cílevědomý návrh na uskutečnění určité inovace v daných termínech zahájení a ukončení.“* [11] Z uvedených poznatků lze odvodit charakteristické znaky projektu:

- je vymezen jeho začátek a konec,
- stanovuje konkrétní cíle,
- definuje strategii vedoucí k dosažení stanovených cílů,
- má omezené zdroje a náklady,
- určuje očekávané přínosy z realizace (konkurenční výhoda, zvýšení zisku atd.),
- je jedinečný a neopakovatelný (nemá vzor v minulosti a ani v budoucnosti se nebude přesně opakovat, každý podobný projekt je v něčem odlišný),
- zpravidla se na jeho realizaci podílí jiný tým specialistů. [10, 11]

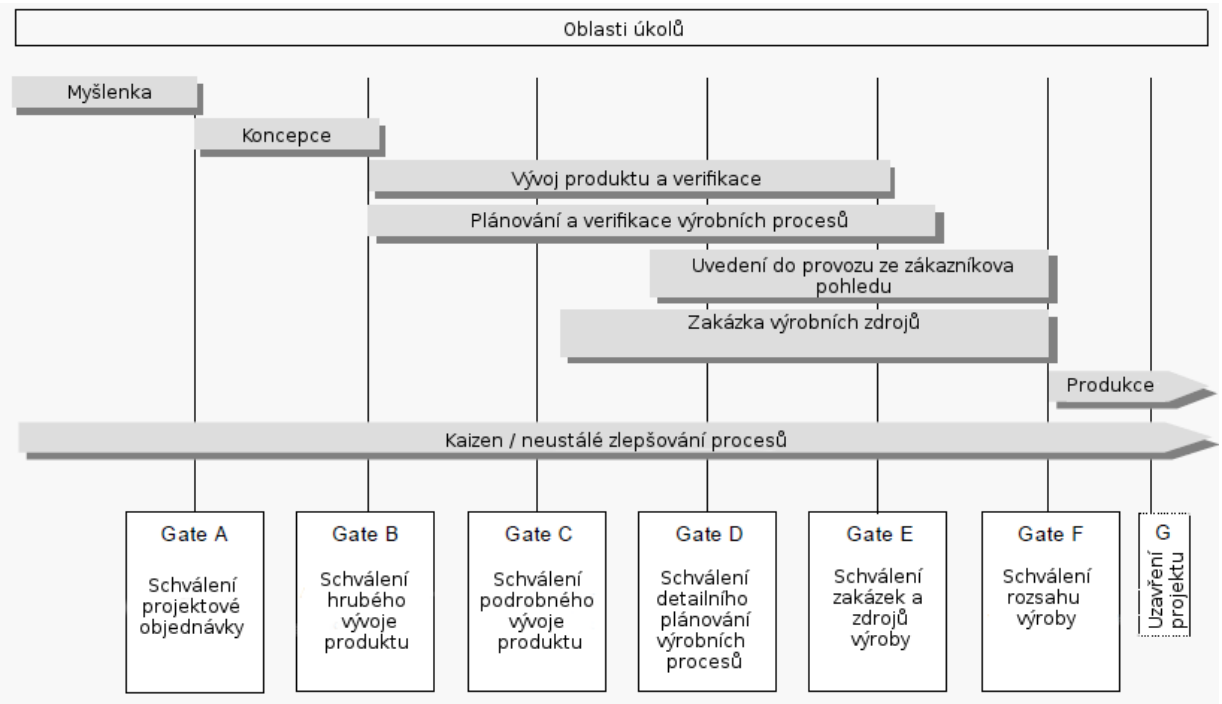
Ve společnosti ZF PLZ jsou rozlišovány tři druhy projektů. Podle toho, o jaký typ projektu se jedná, je stanovena míra zodpovědnosti během realizace projektu a jsou určeny dokumenty, které se budou používat.

- Rozšířené pracoviště (tzv. extended workbench) – jedná se o přijímání a plnění dílčích úkolů od mateřské společnosti zpravidla během jednoho procesu ve V-modelu. Po splnění zadání je výsledek poslán zpět zákazníkovi. Zodpovědnost je pouze za odvedenou práci. V porovnání s ostatními typy projektů je zde zodpovědnost nejmenší (myšleno v kontextu realizace celého projektu).
- Sdílený projekt (tzv. sharing project) – část softwaru je vytvářena v ZF PLZ a část jiným

dodavatelem či zákazníkem. Jasně musí být určeno, kde se budou jednotlivé procesy provádět, případně v jaké míře a kdo za ně bude zodpovědný, aby se předešlo duplicitě plnění úkolů. Velkou roli zde hraje komunikace mezi oběma stranami.

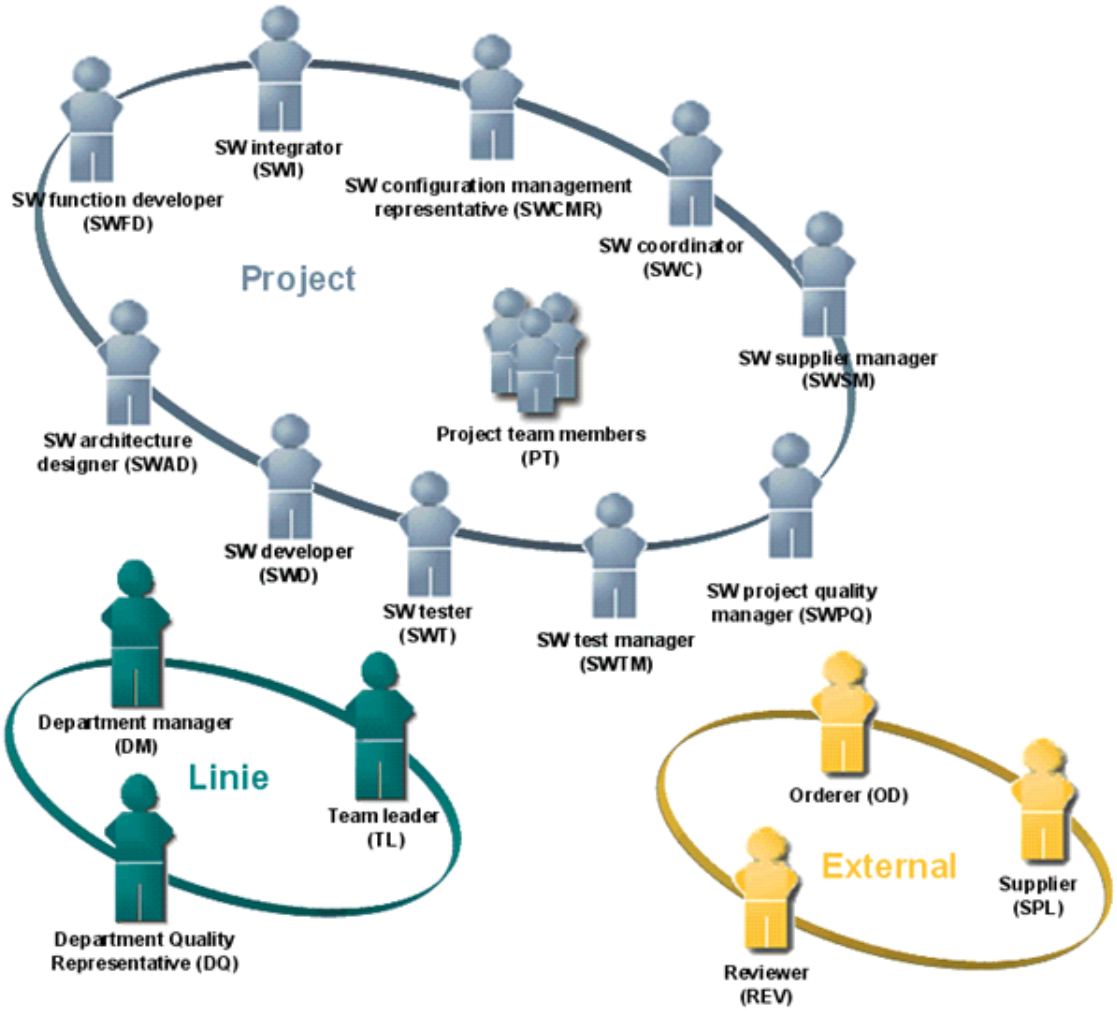
- Místní projekt (tzv. local project) – jde o plnou realizaci projektu ve společnosti. Obvykle jsou prováděny všechny procesy plzeňského V-modelu. Tento typ projektu je zatěžován největší mírou zodpovědnosti.

Projekt je rozdělen do několika kompaktních celků, od definování počáteční fáze při samotném vzniku projektu až po jeho formální ukončení. Určení všech fází projektu přispívá zachovávat jednotné standardy a přehledné sledování průběhu jednotlivých projektů. Konkrétní projektové fáze vývoje produktu ve společnosti ZF PLZ jsou od myšlenky až po výrobu znázorněny na obr. 2.6. Během plnění projektových činností je zajišťováno neustálé zlepšování procesů pomocí malých změn. Při dosažení určité vyspělosti projektu musí proběhnout formální schvalovací procedura, která na základě splnění minima stanovených podmínek povolí vyvíjet produkt na vyšším stupni zpracování. Tento přístup představuje kontrolní mechanismus pro zajištění kvality konečného produktu. Kontrolní bod vývoje produktu je nazván „gate“ (brána). Fázové přechody jsou určeny milníky (tzv. milestones), což jsou významné termíny, které stanovují, do kdy má být daný výstup hotov. [29]




*Obr. 2.6: Fáze projektu při vývoji produktu*

Vzhledem k tomu, že je řízení projektu závislé na aktivitách celého projektového týmu, je nutné mít k dispozici přehled rolí, popsat vztahy mezi těmito rolemi a určit každé roli odpovědnost za splnění dílčích úkolů. Jednotlivé role jsou pak přiřazeny účastníkům projektu, čímž jsou zároveň vymezeny jejich pracovní povinnosti. Každý může vykonávat více rolí najednou, ale nesmí dojít ke střetu zájmů, např. softwarový koordinátor nemůže být zároveň i projektový manažer kvality, neboť by tím byla ovlivněna nezávislost a nestrannost posuzování kvality výrobku. Konkrétní přehled rolí v rámci projektu je zobrazen na obr. 2.7.




*Obr. 2.7: Přehled rolí (převzato z [29])*

- Softwarový funkční vývojář (software function developer – SWFD) se zabývá analýzou jednotlivých požadavků, provádí dle požadavků patřičné úpravy v softwarových funkcích, které pak otestuje. Jeho práce končí, pokud při testování nejsou nalezeny žádné chyby.

- Softwarový integrátor (software integrator – SWI) má za úkol sloučit dílčí části programu a zkontrolovat jejich vzájemnou spolupráci integračními testy.
- Zástupce managementu konfigurace softwaru (software configuration management representative – SWCMR) stanovuje postupy na řízení konfigurace.
- Softwarový koordinátor (software coordinator – SWC) kontroluje včasné dokončení projektových úkolů a také, zda byly naplněny cíle definované v projektovém plánu.
- Softwarový projektový manažer kvality (software project quality manager – SWPQ) zajišťuje nezávislou kontrolu kvality pracovních výstupů v rámci projektu.
- Manažer testování softwaru (software test manager – SWTM) je zodpovědný za včasné provedení všech testů a jednotlivé softwarové testy koordinuje a kontroluje.
- Tester softwaru (software tester – SWT) vytváří na základě specifikace požadavků funkční testy, které poté provádí. Cílem jeho práce je odhalit co nejvíce chyb v softwaru.
- Softwarový vývojář (software developer – SWD) vytváří detailní design produktu, implementaci a kontroluje zdrojový kód unitovými testy a modulovými testy.
- Softwarový návrhář architektury (software architecture designer – SWAD) se zabývá návrhem architektury jednotlivých komponent a rozhraním mezi nimi.
- Člen projektového týmu (Project team members – PT) se podílí na realizaci projektu a je zodpovědný za plnění dílčích projektových aktivit.
- Manažer oddělení (department manager – DM) řídí a kontroluje své podřízené, zastřešuje chod svého oddělení, podílí se na schvalovacích procedurách důležitých dokumentů.
- Zástupce oddělení kvality (department quality representative – DQ) sleduje plnění cílů kvality v projektech celého oddělení.
- Vedoucí týmu (team leader – TL) zajišťuje rozdělení dílčích projektových úkolů svým podřízeným.
- Zákazník (orderer – OD) zadává projekt dodavateli a očekává splnění všech svých požadavků.
- Dodavatel (supplier – SPL) poskytuje výrobky či služby svému odběrateli.
- Kontrolor (reviewer – REV) přezkoumává dle svého oprávnění správnost pracovních výstupů svých kolegů. [29]

Cílem procesu řízení projektu je identifikovat, stanovit, plánovat, koordinovat a monitorovat aktivity, úkoly a zdroje potřebné k výrobě produktu v souvislosti s požadavky projektu a omezeními. [13] V automobilovém průmyslu obvykle zahrnují projekty vývoj hardwaru, softwaru a mechanických prvků. To se odráží v organizaci projektu, neboť je rozčleněn do několika dílčích projektů, na kterých pracují různé týmy odborníků.

Při vytváření projektu musí být stanoven jeho přesný rozsah a dále musí být potvrzeno, že definovaných cílů lze dosáhnout s dostupnými zdroji a omezeními. Vymezení rozsahu projektu se skládá z několika částí. Nejprve je zdůvodněn účel projektu. Dále jsou sestaveny cíle projektu, včetně měřitelných kritérií, která posuzují dosažení cíle, také jsou popsány produkty, které mají být vyvinuty a je zpracován přehled všech výstupů, které mají být vytvořeny. Rozsah projektu se může časem změnit. Důvodem je např. přidání nových funkcí, které nebyly známy na začátku projektu. Rozsah, cíle i zdroje projektu jsou dokumentovány v projektovém plánu.

V rámci procesu projektového řízení je nutné identifikovat požadované kvalifikační dovednosti, které jsou nezbytné pro realizaci projektu. Na základě srovnání s kvalifikací zaměstnanců, musí být jejich případné chybějící znalosti doplněny.

Pokrok projektu je sledován skrz tzv. projektový status. Jedná se o odpovězení na předem připravené otázky a jejich následné hodnocení, které je založeno na principu semaforu. Zelená barva signalizuje, že je vše v pořádku, oranžová barva značí výskyt drobného problému a červená barva upozorňuje na závažné nedostatky, které je nutno co nejdříve odstranit. Projektový status se aktualizuje v pravidelných časových intervalech. Pokud jsou během sledování pokroku nalezeny odchylky proti stanovenému plánu a hrozí, že cíle projektu nebudou moci být splněny, musí být zahájeno nápravné opatření (např. změna počtu členů týmu, upravený rozsah činností či otevřený dialog se zákazníkem o zpoždění). [29]

Pro přehledné plánování a řízení projektů, sledování termínů, správu úkolů nebo přiřazování zdrojů je používán nástroj Microsoft Project. Výstupní data mohou být zobrazena v různých formách např. Ganttův diagram, síťový diagram, kalendář.

## 2.13 *Monitorování dodavatele*

Během procesu „Monitorování dodavatele“ je vybrán vhodný externí dodavatel a poté se sleduje a kontroluje plnění požadavků, které s ním byly ujednány. V Plzni tento proces není

v současné době vykonávaný. Společnost zpracovává všechny projekty sama a zatím nevyužívá dodavatelských služeb. Proces je zaveden především v Německu, kdy mateřská společnost přiřazuje zakázky ostatním pobočkám, které se stávají jejím interním dodavatelem. Obecně lze říci, že dodavatel je společnost nebo její část, která na základě smlouvy se zadavatelem projektu poskytuje realizační zdroje a know how k dosažení nadefinovaných cílů projektu.

Na začátku spolupráce mezi společností a dodavatelem je nutné dohodnout např. cenu, termíny, rozsah práce, společné postupy, odpovědnosti, typ a četnost společných aktivit a zprávy o stavu produktu a jeho hodnocení. Velký důraz je kladen na komunikaci, která musí být udržována během celého projektu, aby bylo zajištěno, že dodavatel rozumí ustanoveným požadavkům a veškeré práce se provádí v souladu s plánem. Pravidelně se kontroluje pokrok v projektu z hlediska plnění plánu, kvality, nákladů a sledují se problémy a rizika. Pokud se jedná o závažný problém (např. potíže s kvalitou, posun termínů) musí být prodiskutován oběma stranami. [13, 29]

## 2.14 *Řízení konfigurace*

Cílem procesu „Řízení konfigurace“ je stanovit a udržovat integritu všech pracovních produktů a poskytnout je k dispozici účastníkům projektu. Řízení konfigurace umožňuje vidět rozdíly mezi verzemi dokumentů, či zdrojového kódu podle časové posloupnosti vytvoření. Dále je možné provádět změny v dokumentech, aniž by došlo k jejich ohrožení, protože se lze kdykoliv vrátit k původní verzi. Také zde existuje možnost paralelní práce mnoha lidí na téže verzi dokumentu. Všechny dokumenty jsou uloženy na tzv. VOBu (Versioned Object Base). Jedná se o verzovací systém, který ukládá jednotlivé soubory a informace o nich (např. jméno autora, datum vytvoření) a díky tomu umožňuje dohledatelnost změn historie dokumentu. [13, 29]

Hlavním používaným nástrojem pro řízení konfigurace je ClearCase vyvinutý od společnosti IBM Rational. Poskytuje propracovanou správu verzí a pracovního prostoru a umožňuje podporu souběžného vývoje. Pomocí nástroje lze bezpečně opravovat dříve dodané produkty.

## 2.15 *Řízení požadavku na změnu*

Účelem procesu „Řízení požadavku na změnu“ je řídit, sledovat a kontrolovat všechny

požadavky, které se týkají změn. Tyto požadavky jsou shromažďovány stanoveným způsobem, který umožňuje sledovat jejich pokrok ve zpracování. Pro společnost je to důležité, neboť žádný projekt není beze změn. Naopak, mnoho projektů je doslova zaplaveno změnami. Hlavními důvody jsou:

- změny způsobené odstraňováním vad (proces řízení problémů),
- změny, které pochází od zákazníka.

Proces řízení požadavku na změnu začíná s žádostí o změnu a končí jejím zpracováním a uzavřením nebo zrušením změny. Vše je prováděno v nástroji ClearQuest. Každá fáze zpracování změny je označena novým stavem, jehož prostřednictvím jsou účastníci projektu informováni o průběhu realizace. Každá změna je podrobena analýzou proveditelnosti, možných rizik a je posuzován dopad změn na produkt. Požadavkům na změny je přiřazena priorita, která označuje stupeň naléhavosti pro implementaci změny. Před konečným uzavřením požadavku na změnu musí být zkontrolováno, zda byla změna správně provedena v souladu s cíli. [13, 29]

## 2.16 *Zajištění kvality*

Proces „Zajištění kvality“ poskytuje nezávislou kontrolu pracovních produktů a procesů a zjišťuje, zda jsou v souladu s nadefinovanými cíli a plány. Pojem kvalita lze obecně definovat jako „*stupeň splnění požadavků souborem inherentních znaků*“. [14] Za inherentní znaky jsou považovány parametry výrobku, které přímo podmiňují funkci, pro kterou byl produkt navržen. Zjednodušeně řečeno, kvalita znamená, že se vrací zákazník, nikoliv výrobek. [14]

Kvalita softwaru je zajišťována a sledována od počátku zahájení práce na projektu. Zajištění kvality nezahrnuje tedy pouze testování softwaru, ale i stanovení procesů a metod, které jsou potřebné pro správné určení požadavků a vývoj produktu. Zajištění kvality spočívá i v kontrole dodržování procesů a v kontrole jejich výstupů.
Software má z hlediska kvality specifické vlastnosti:

- povaha programů je nehmotná,
- programy jsou složité a procesy na jejich tvorbu také,
- potřeba odpovídající úrovně znalostí,
- nelze dokázat, že program je bez chyb (testování pouze prokazuje, že program určité

chyby nemá). [31]

Každý projekt je jednou za rok hodnocen externím auditorem a je určeno, na jaké úrovni jsou vykonávané procesy dle normy Automotive SPICE. Celkem je rozlišováno 6 úrovní způsobilosti.

- Úroveň 0 – proces není vykonáván nebo účel procesu není splněn.
- Úroveň 1 – základní postupy jsou v procesu zavedeny a definovaných výsledků je dosaženo nejednotnými postupy.
- Úroveň 2 – výkonnost procesu je plánována a sledována. Pracovní produkty jsou pod správou konfigurace, je zajištěna kvalita. Proces je řízen.
- Úroveň 3 – je stanoven standardní proces a platí v rámci celé organizace. Projekty používají přizpůsobenou verzi standardního procesu. Proces je schopen dosáhnout definovaných výsledků procesu.
- Úroveň 4 – při plnění stanoveného procesu se provádí detailní analýzy vedoucí ke kvantitativnímu pochopení výkonnosti procesu.
- Úroveň 5 – proces je neustále zlepšován, nové inovativní přístupy a metody nahrazují méně účinné postupy a vedou k lepšímu dosažení předem určeného cíle. [13]

Cílem je ve společnosti ZF PLZ dosáhnout a udržovat procesy v jednotlivých projektech na druhé nebo třetí úrovni způsobilosti.

V každém projektu je sestaven plán kvality, ve kterém jsou stanoveny cíle kvality projektu, metody a nástroje, které se používají k jejich dosažení. O tom, v jakém aktuálním stavu se nachází projekt z hlediska kvality, informuje status kvality. Ke sledování kvality a jejímu vyhodnocování se využívají různé metriky. Metrika představuje číselný údaj sledovaného jevu a musí umožňovat srovnání s určenou referenční hodnotou. Konkrétní metrikou může být porovnání počtu nalezených chyb v softwaru za určité časové období s předem pevně stanoveným maximálním počtem chyb. [29]

Význam zavádění a zlepšování kvality spočívá v zajištění spokojenosti a loajality zákazníka (čímž je upevňována pozice na trhu), dále je zvyšována účinnost a produktivita

procesů (tím je dosaženo lepšího hospodářského výsledku) a v neposlední řadě dochází ke zvýšení profesionality zaměstnanců ve společnosti.

Pro popis a řízení procesů ale i pro shromáždění dokumentace ke konkrétním projektům slouží nástroj Stages. Podrobněji je tento nástroj popsán v kapitole 3.

## 2.17 *Řízení problému*

Proces „Řízení problému“ zajišťuje, že jsou všechny odhalené problémy identifikovány, analyzovány, sledovány a kontrolovány strukturovaným dohledatelným způsobem. K určení zlepšení nebo zhoršení řízení problému v projektu pomáhá shromažďování statistických údajů, např. o době setrvání problému v určitém stavu či odvození trendů. Proces lze aplikovat na problémy a vady všeho druhu a je úzce spjat s procesem řízení požadavku na změnu, ve kterém je chyba odstraňována. Proces řízení problému tedy poskytuje vstupy procesu řízení požadavku na změnu. Po odstranění chyby je v procesu řízení problému ověřováno, zda byla chyba skutečně opravena a k incidentům již nebude docházet.

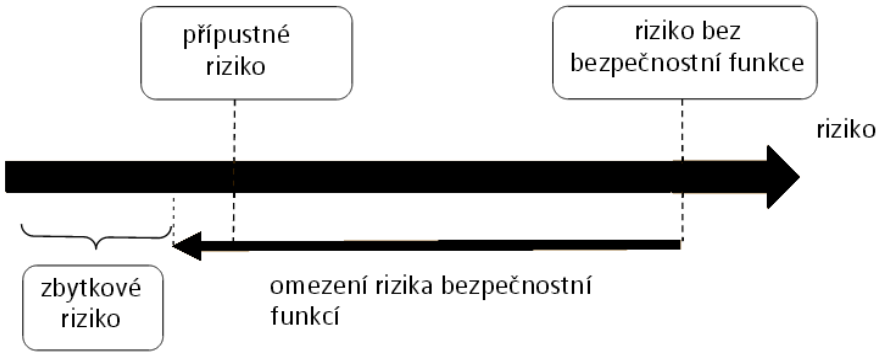
Každý problém musí být zaznamenán, jednoznačně označen (pomocí ID) a detailně popsán. Čím je popis podrobnější, tím je jednodušší analýza jeho příčin. O tom, v jakém pořadí se budou jednotlivé problémy řešit, rozhoduje jejich stanovená priorita. Kritéria pro stanovení priorit mohou být např. kritičnost, naléhavost či očekávané dopady. [13, 29]

Nástroj využívaný pro správu problému je Omnitracker. Nástroj poskytuje uživatelům moderní grafické rozhraní a lze k němu přistupovat pomocí webového klienta. Všechny informace jsou v něm řízeny podle definovaných postupů.

## 2.18 *Funkční bezpečnost*

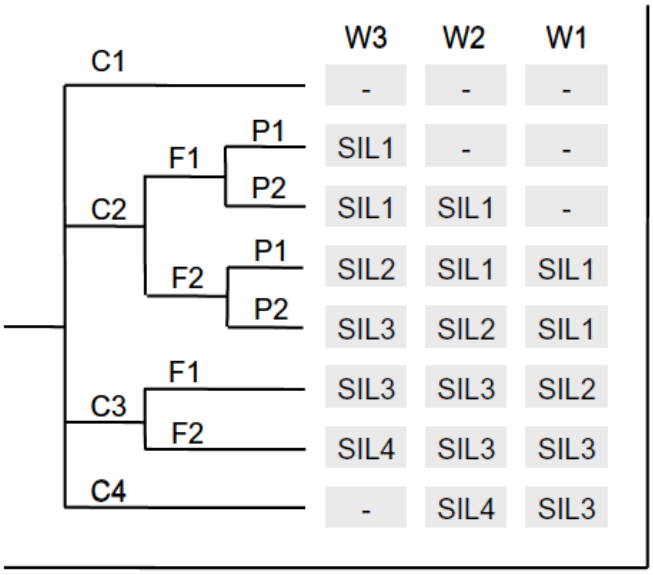
Proces „Funkční bezpečnost“ se zaměřuje na identifikaci potenciálně nebezpečných závad nebo poruch, které mohou způsobit poškození zdraví, majetku, životního prostředí a v nejhorším případě ztrátu života. Cílem je odhalit všechna nepřípustná rizika a transformovat je na přípustná. Docílit absolutně nulového rizika nelze, viz obr. 2.8. Riziko existuje všude, kde může chybná funkce zapříčinit újmu. Hlavními klíčovými prvky pro ohodnocení rizika je pravděpodobnost vzniku nebezpečné události a očekávaná závažnost zranění a rozsah škod. Funkční bezpečnost je možné definovat jako část celkové bezpečnosti, která závisí na správné

činnosti zařízení a řídicích systémů zajišťujících bezpečnost. Zavedením ochranné funkce je snížena pravděpodobnost výskytu nebezpečných selhání. Funkční bezpečnostní prvky tvoří nedílnou součást každé fáze vývoje produktu.




*Obr. 2.8: Omezení rizika*

Problematika funkční bezpečnosti je řešena ve standardu IEC 61508. Tento standard zavádí úrovně integrity bezpečnosti (SIL). Jedná se o kategorie, které stanovují, jak často může v systému nastat nebezpečná porucha s rizikem vzniku nebezpečné události, která může mít za následek fyzické nebo materiální ztráty. Existují čtyři úrovně SIL, kde SIL 4 znamená nejvyšší a SIL 1 nejnižší úroveň integrity bezpečnosti. Pro úplnost bývá někdy zavedena i SIL 0, což je úroveň bez potřeby na bezpečnost. Metoda používaná k přiřazení SIL k jednotlivým požadavkům je např. prostřednictvím grafu rizik, viz obr. 2.9. Z obrázku je patrné, že stejná úroveň SIL může být klasifikována na základě odlišných kritérií. Těmito kritérii je rozsah poškození, frekvence ohrožení, možnost vyhnutí se ohrožení a pravděpodobnost vzniku poruchy. [13, 25]




Rozsah poškození

C1 - lehké zranění osob; malé poškození prostředí
C2 - těžké zranění jedné nebo více osob; smrt jedné osoby; větší poškození prostředí
C3 - smrt více osob; velké poškození prostředí
C4 - katastrofa; smrt mnoha osob

Frekvence ohrožení

F1 - zřídka
F2 - často nebo trvale

Možnosti vyhnutí se

P1 - možné za určených podmínek
P2 - téměř nemožné

Pravděpodobnost poruchy

W1 - velmi malá
W2 - malá
W3 - relativně velká

*Obr. 2.9: Určování SIL podle grafu rizik*

Norma, která se zabývá funkční bezpečností přímo pro vývoj softwaru elektrických nebo elektronických systémů v automobilovém průmyslu a vychází z výše jmenovaného standardu, je ISO 26262. [35]

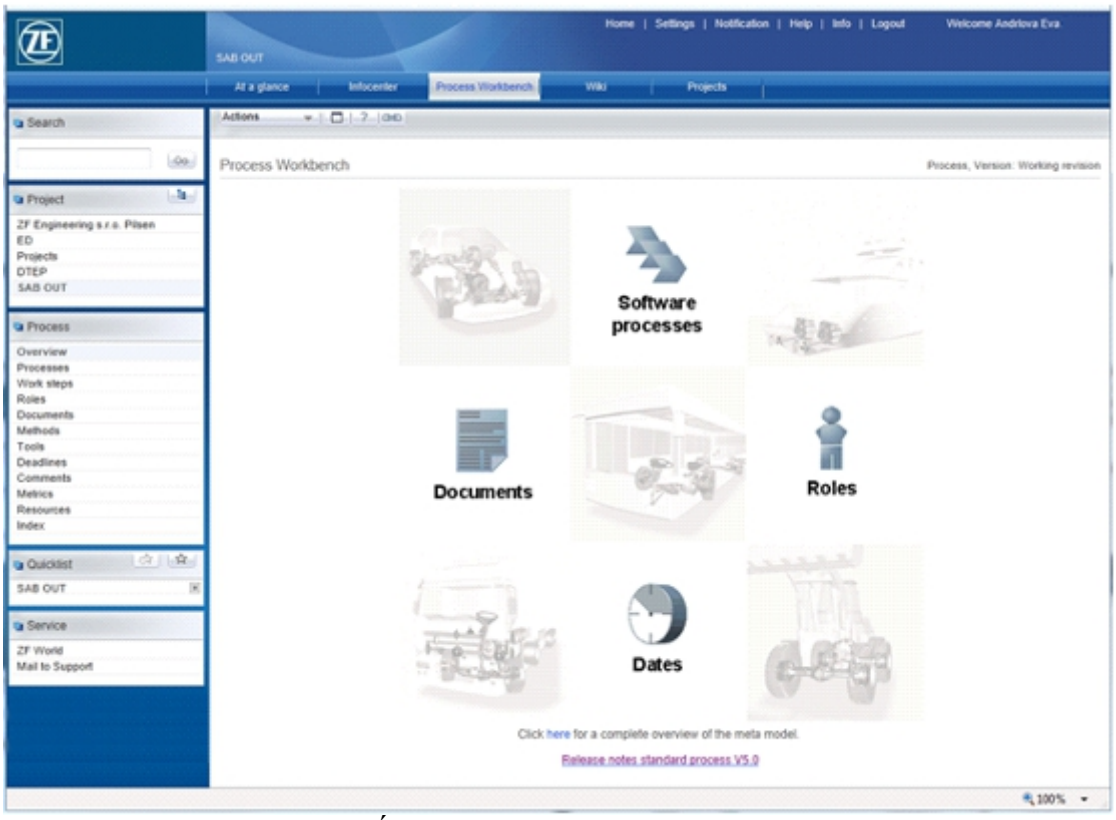
# 3 Nástroj pro popis procesů - Stages

Tato kapitola je zaměřena na obecné představení nástroje používaného pro řízení a popis jednotlivých procesů a používaných metod ve společnosti ZF PLZ v oddělení elektroniky. Nástroj je nazván Stages (v současné době je v Plzni k dispozici verze 5.1.21.0) a je vyvinut od společnosti Method Park Software AG. Popis procesů je realizován psaným textem i grafickým znázorněním pro lepší pochopení a zvýšení přehlednosti. Díky zdokumentování všech prováděných činností je know-how uchováváno neustále ve firmě a není závislé na jedincích. Know-how jsou informační a technologické předpoklady a znalosti pro konkrétní aktivity. Ve Stages jsou ke každému projektu shromážděny relevantní dokumenty z různých zdrojů a tím je zajištěn k těmto dokumentům jednotný přístup. Díky internímu webovému rozhraní, jsou informace dostupné z několika lokalit (např. Plzeň, Friedrichshafen, Schweinfurt).

Je zde nadefinován standardní proces (viz kapitola 2), který se na míru upravuje podle konkrétního projektu. Nehodící se procesy jsou vynechány a neprovádějí se. Tato úprava je označována jako „tailorování". Prostředí ve Stages je interaktivní, což významně usnadňuje práci uživatelům a šetří se tím jejich čas.

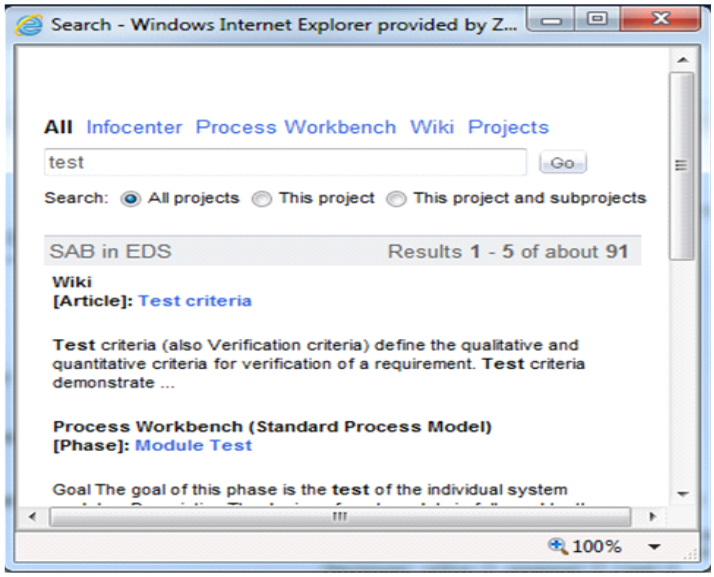
Nástroj umožňuje zobrazovat různé množství informací. Záleží na tom, zda je uživatel přihlášený či nikoliv a jaké jsou mu přiřazeny role. Každá role má přidělena určitá práva, která stanovují, k jakým datům má uživatel přístup a jak s nimi může nakládat (např. čtení, editace). Obecné charakteristiky prováděných činností jsou k dispozici vždy a k údajům o projektu mají přístup jen přihlášení uživatelé s přidělenými právy, resp. rolemi. Nástroj je představen z pohledu role softwarového projektového manažera kvality.

Na obr. 3.1 je možno vidět úvodní stránku konkrétního uživatele. V levé části Stages se nachází panel nabídek skládající se z několika částí, jejichž počet je závislý na vybrané sekci v záhlaví. Sekce „Process Workbench" má v navigačním menu k dispozici pět oblastí.




*Obr. 3.1: Úvodní stránka ve Stages po přihlášení*

První oblast „Search" souží k vyhledávání informací pomocí klíčových slov. Výsledky jsou zobrazeny v samostatném okně, ve kterém je možno dodatečně nastavit, zda má být hledání uskutečněno pouze v rámci jednoho projektu nebo napříč všemi projekty a také lze zvolit vyhledávání v konkrétní sekci, viz obr. 3.2.




*Obr. 3.2: Výsledky vyhledávání podle klíčového slova*

„Project" slouží k vyhledání daného projektu. Pokud je kliknuto na ikonu v hlavičce této oblasti, je otevřeno nové okno, kde je zobrazena stromová struktura pro přehlednější hledání.

Třetí oblast „Process" je zaměřena na snadné přepínání mezi popisy procesů, pracovních kroků, nástrojů, rolí, metod atd. kdykoliv během práce.

„Quicklist" umožňuje rychle přepínat z jednoho projektu do druhého. Nový projekt lze přidat do osobního seznamu kliknutím na symbol hvězda. Všechny vybrané projekty se zobrazí ve stromové struktuře projektu a poté lze aktivovat políčka u projektů, které se mají přidat do seznamu a deaktivovati políčka u projektů, které mají být z quicklistu odstraněny.

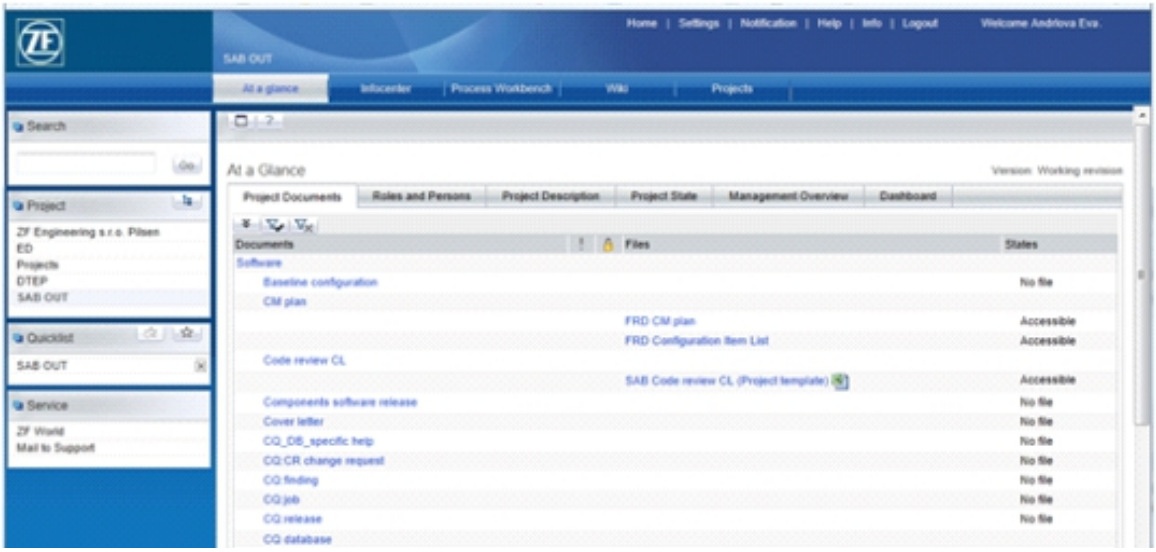
V záložce „Service" se lze připojit na domovské stránky společnosti ZF a je tady také možnost poslat e-mail webmasterovi v případě nějakého problému či dotazu.

Stages používá dvě oblasti záhlaví. Jedna z nich obsahuje logo společnosti, jméno přihlášené osoby a standardní akce jako vrátit se na úvodní stránku příkazem "Home", dále se může zvolit nastavení „Setting" (např. volba úvodní stránky, jazyku, změna hesla atd.), oznámení „Notification", nápověda „Help", informace „Info" (o verzi programu, licenci atd.) a odhlášení „Logout". V druhé oblasti záhlaví se vyskytují všechny přístupné sekce, které podávají informace o vybraném projektu. Pod nimi je lišta s ikonami, které umožňují různé akce. Přehled funkcí jednotlivých ikon je zpracován v tab. 1.

*Tab. 1: Funkce jednotlivých ikon*

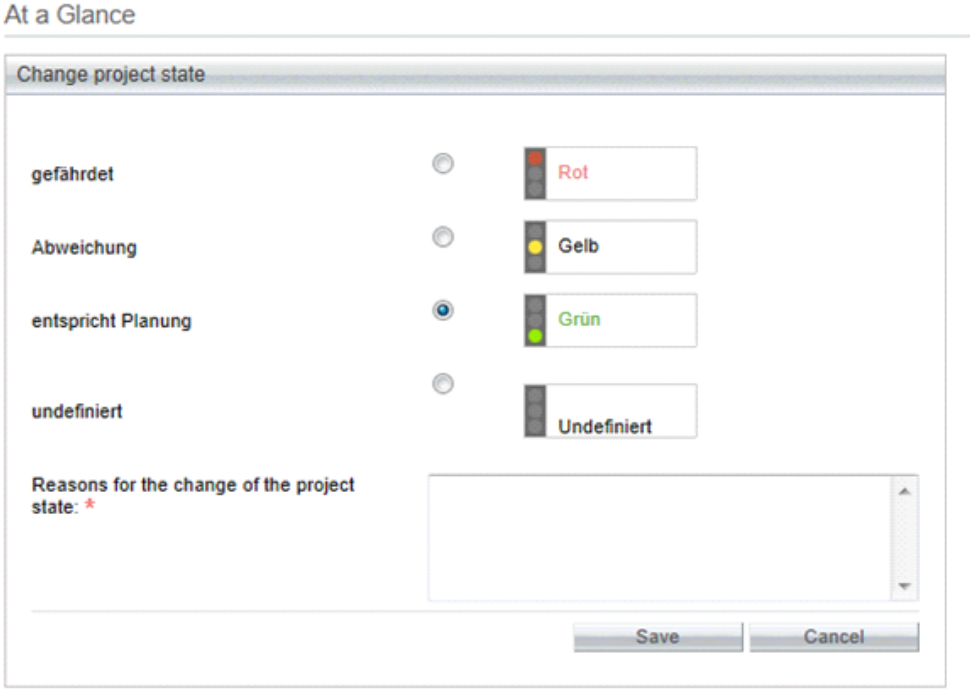
| Ikona | Funkce |
|---|---|
|  | Nápověda |
|  | Generování odkazu na aktuální stránku |
|  | Tisk aktuální a všech souvisejících stránek |
|  | Tisk aktuální stránky |
|  | Zobrazení na celou obrazovku |

Sekce „At a glance" předkládá uživateli přehled o všech existujících dokumentech (obr. 3.3) a přiřazených rolí ke konkrétním osobám. Také poskytuje stručný popis projektu a jeho stav hodnocený metodou semaforu, viz obr. 3.4 a obr. 3.5. Dále lze zobrazit stavy všech projektů včetně data poslední aktualizace a zjistit informace o projektu (např. grafy znázorňující trendy nebo výskyt sledovaného jevu).




*Obr. 3.3: Přehled projektové dokumentace*

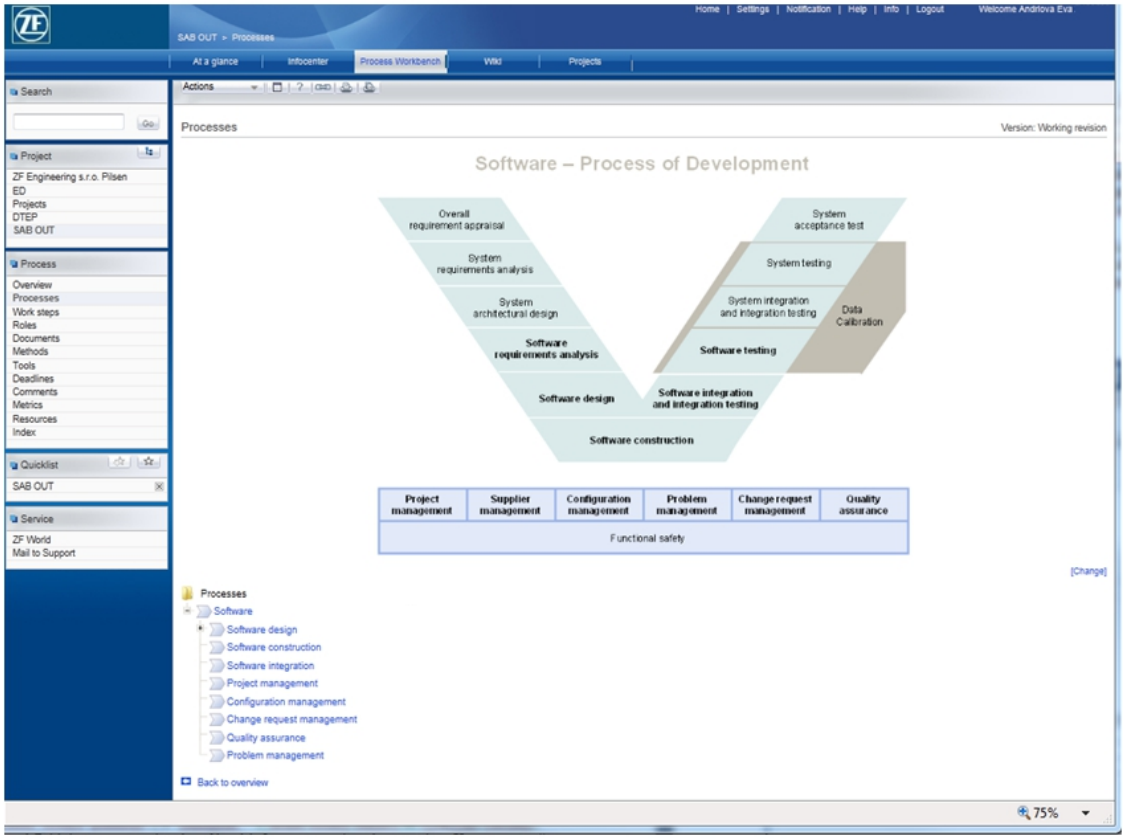

*Obr. 3.4: Přehled stavu projektu*



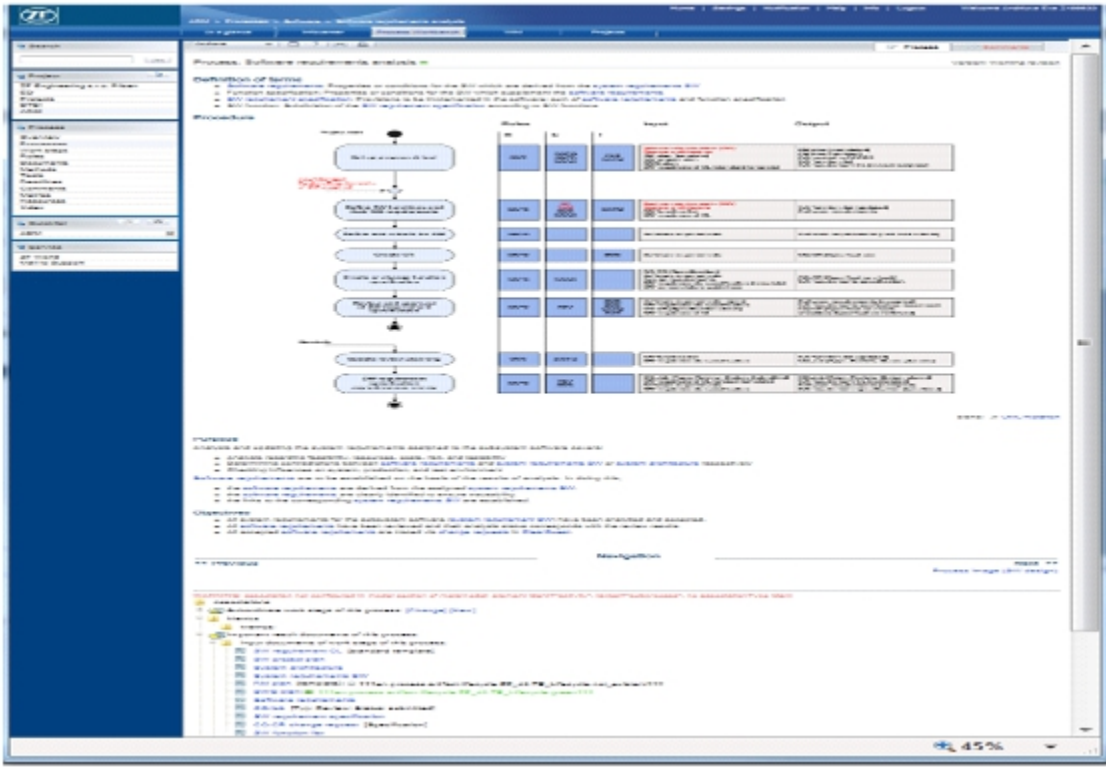

*Obr. 3.5: Nastavení stavu projektu dle metody semafor*

Sekce „Infocenter" shromažďuje výsledky ohledně tailorování daného projektu. Sledují se tím průběžně změny, které byly provedeny.

Sekce „Process workbench" je ve Stages stěžejní. Jsou zde teoreticky popsány všechny role, procesy, pracovní kroky, metody, nástroje atd. Úvodní stránka je na obr. 3.1. Po kliknutí na „Software processes" jsou zobrazeny procesy dvojím způsobem - jednak v grafické podobě V-modelu a níže ve formě stromové struktury, viz obr. 3.6. Po zvolení konkrétního procesu je zobrazena definice procesu a vývojový diagram, viz obr. 3.7. Ke každému pracovnímu kroku jsou přehledně přiřazeny vstupní a výstupní dokumenty a role, které určují, kdo je za vykonání činnosti zodpovědný, kdo má být informován a s kým má být činnost projednána. V další části stránky se vyskytuje navigace umožňující rychlý přechod na následující nebo předchozí proces a seznam odkazů týkajících se daného procesu (např. odkazy na dokumenty).

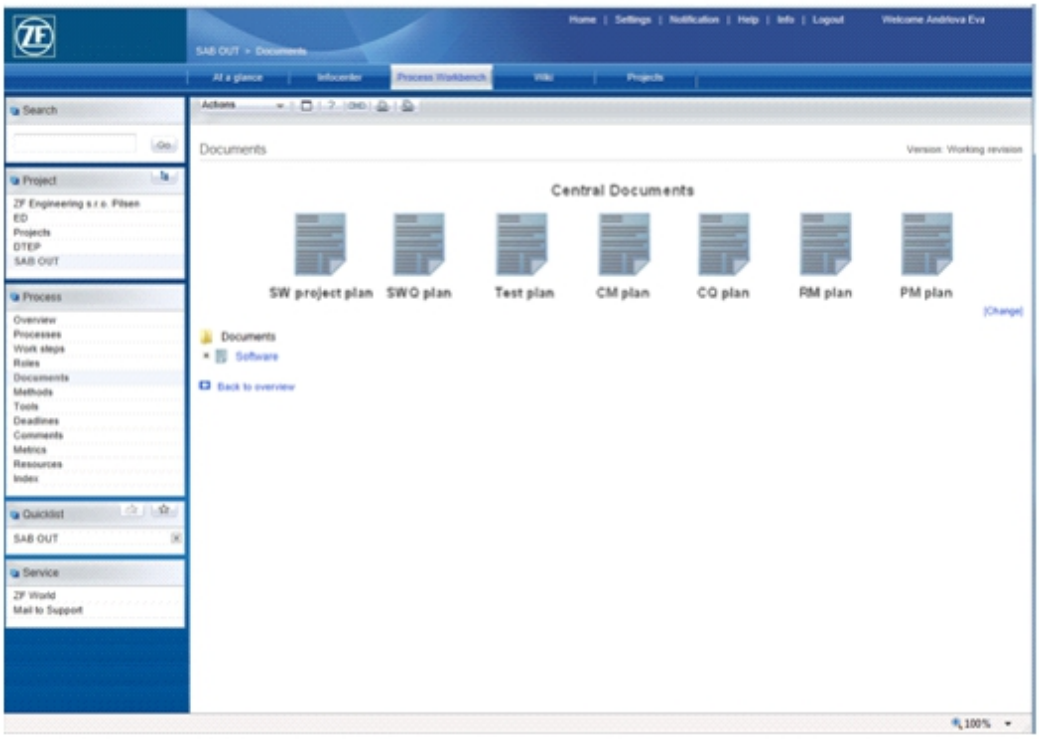



*Obr. 3.6: Přehled procesů*

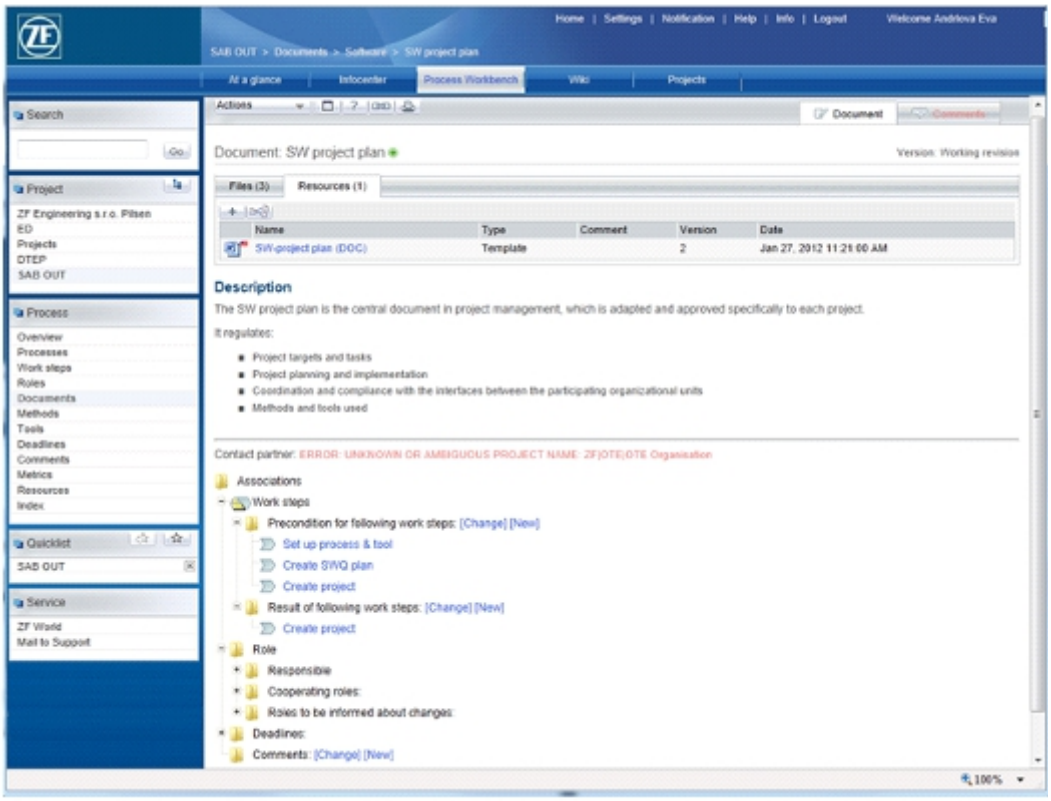


*Obr. 3.7: Popis a grafické znázornění procesu*

Pokud je na úvodní stránce sekce „Process workbench“ kliknuto na „Documents”, je otevřena stránka s grafickým přehledem všech centrálních dokumentů a pod nimi je možnost zobrazit všechny dokumenty pomocí stromové struktury, viz obr. 3.8. Centrální dokumenty jsou základem pro realizaci každého projektu. Po vybrání dokumentu je načtena stránka, viz obr. 3.9. Je zde umístěn popis dokumentu a asociované odkazy. V záložce „Resources” jsou k dispozici šablony, checklisty, vzorové dokumenty včetně informace o jejich verzi a data zveřejnění. Tyto šablony jsou pod správou kvality a slouží jako podklad pro zpracování konkrétního zadání, které je pak uloženo v záložce „File” tvůrcem dokumentu.

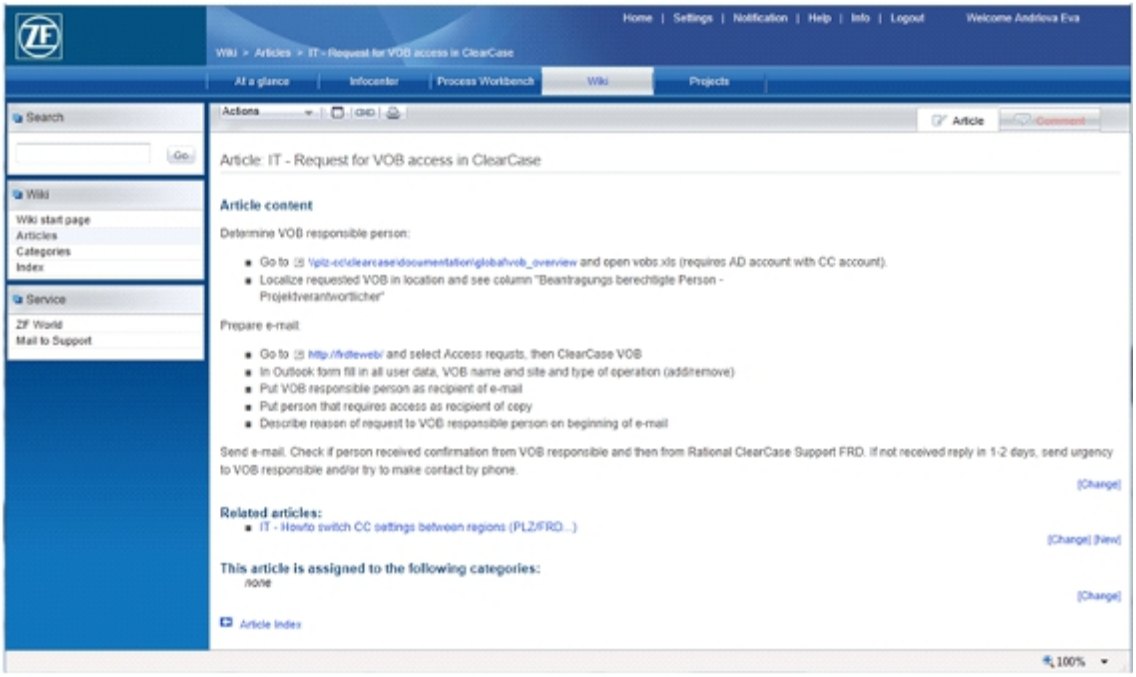



*Obr. 3.8: Přehled centrálních dokumentů*



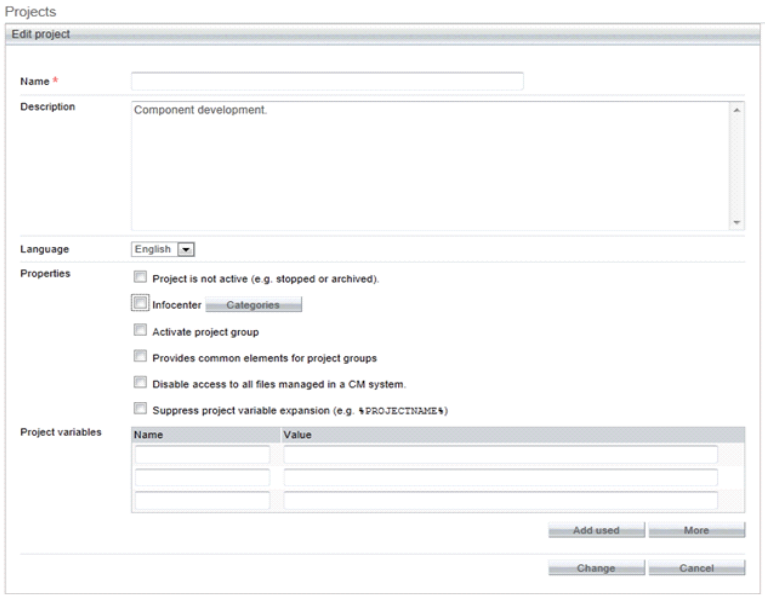

*Obr. 3.9: Popis projektového plánu*

Sekce „Wiki" nabízí vkládání a pročítání odborných článků, které napsaly autorizované osoby, viz obr. 3.10. Každý článek může být prolinkován s podobnými příspěvky a u každého článku mohou zaměstnanci vkládat užitečné komentáře a tím rozšiřovat a prohlubovat znalosti týkající se daného tématu. Články lze třídit dle abecedy, nebo podle tématických kategorií.




*Obr. 3.10: Příklad článku v sekci Wiki*

Sekce „Projects" umožňuje editovat a zobrazovat popis projektu (stejný popis se pak objevuje i v sekci „At a glance" v záložce „Project description"). Podoba editoru s možnými nastavitelnými vlastnostmi včetně jazyka je na obr. 3.11.




*Obr. 3.11: Editor pro popis projektu*

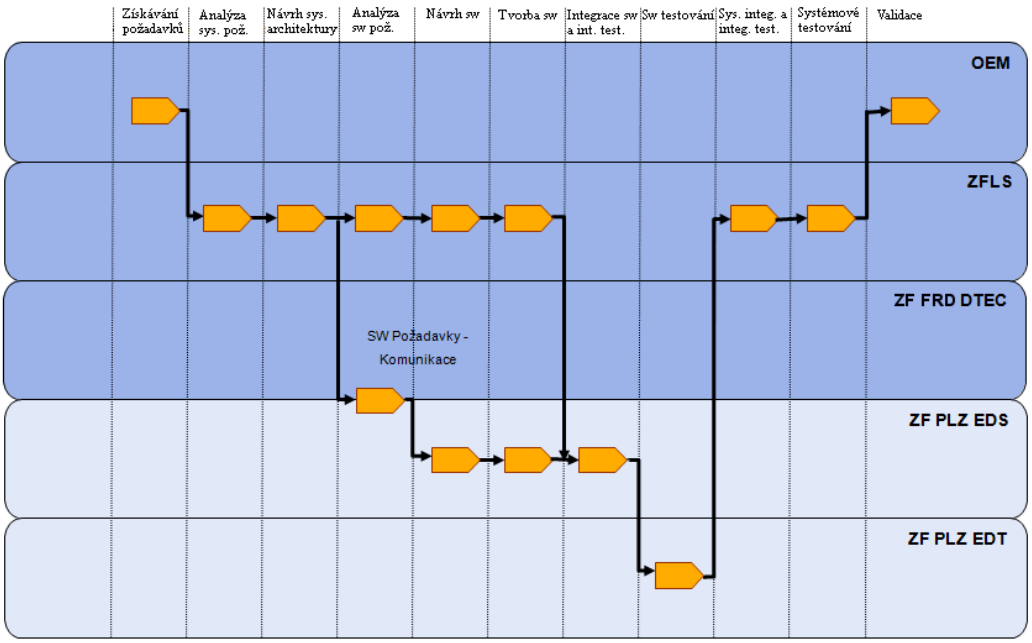
Mezi výhody nástroje Stages se řadí:

- jasná definice a transparentnost procesů,
- přehled o projektu a jeho aktuálním stavu,
- jednoduché a snadné sdílení dokumentů,
- dokumenty na jednom místě,
- vyšší efektivita spolupráce,
- manipulace a náhled dokumentů na základě udělených práv,
- uživatelsky přátelské prostředí,
- přístup pomocí webového prohlížeče,
- přístup z několika lokalit,
- možnost vkládání komentářů.

Za nevýhody může být považována vysoká pořizovací cena, náklady na údržbu, nutnost školení v zacházení s nástrojem a dlouhodobější reakce na změny či zlepšení nástroje. Osoba zodpovědná za Stages pořádá pravidelné schůzky s Method Park, kde jsou vždy projednány zlepšení sesbírané z procesních nálezů během realizace projektů. Změny se projeví v další verzi nástroje.

# 4 Procesní řízení v projektu Honda

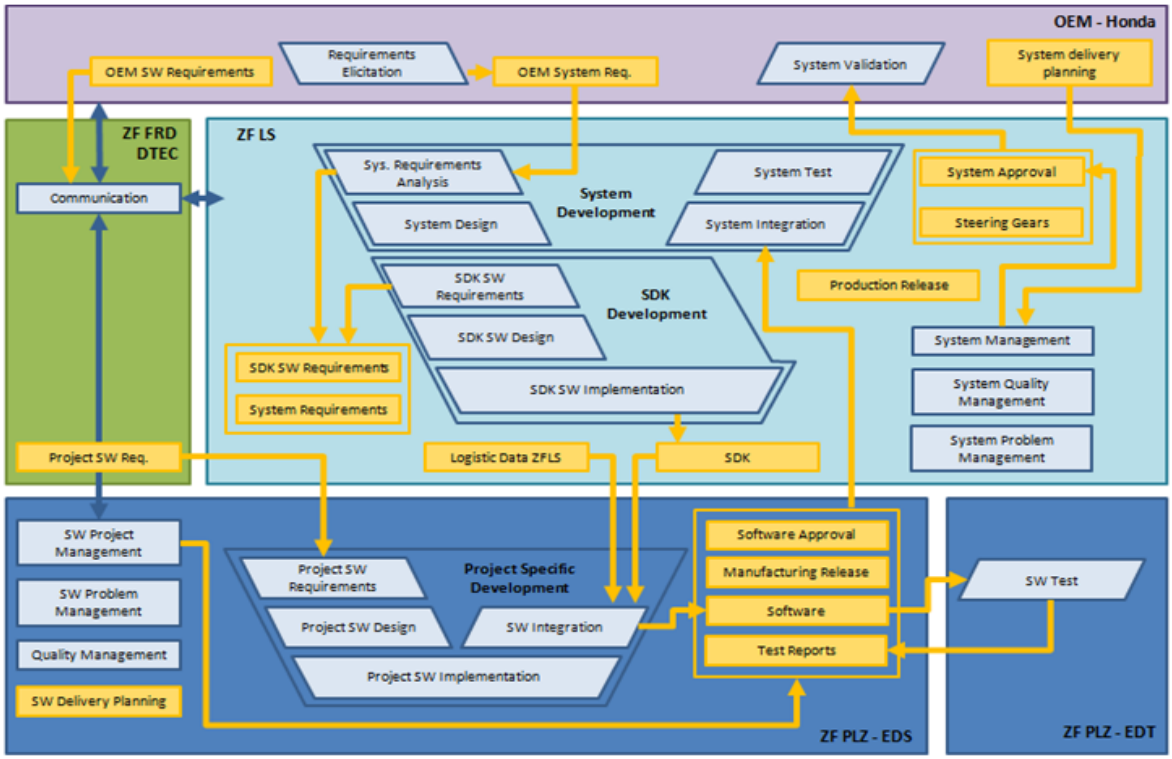
V této kapitole je popsáno procesní řízení v projektu Honda, který se zabývá vývojem softwaru pro posilovače řízení. Z důvodu utajovaných informací, jsou na použitých obrázcích změněny identifikační data. Pojem procesní řízení lze interpretovat jako ujištění, že *„procesy pracují na nejvyšší úrovni jejich potenciálu, vyhledávání příležitostí jejich zlepšení a přenesení těchto příležitostí do reality.“* [36] Projekt Honda je realizován na základě standardních procesů popsaných ve 2.kapitole. Projekt patří do skupiny sharing projektů, u nichž některé pracovní kroky nebo celé procesy z V-modelu softwarového vývoje mohou být vykonávány jinde, např. u zákazníka, a některé mohou být sdíleny. Projektu se účastní pět obchodních partnerů. Na obr. 4.1. je přehledně znázorněno, kdo jaké procesy provádí a jejich vzájemná posloupnost či provázanost. ZF PLZ EDT a ZF PLZ EDS jsou oddělení v plzeňské pobočce, ZF FRD DTEC a ZFLS jsou pracoviště v Německu, OEM představuje finálního výrobce.




*Obr. 4.1: Přehled provádění procesů u jednotlivých partnerů*

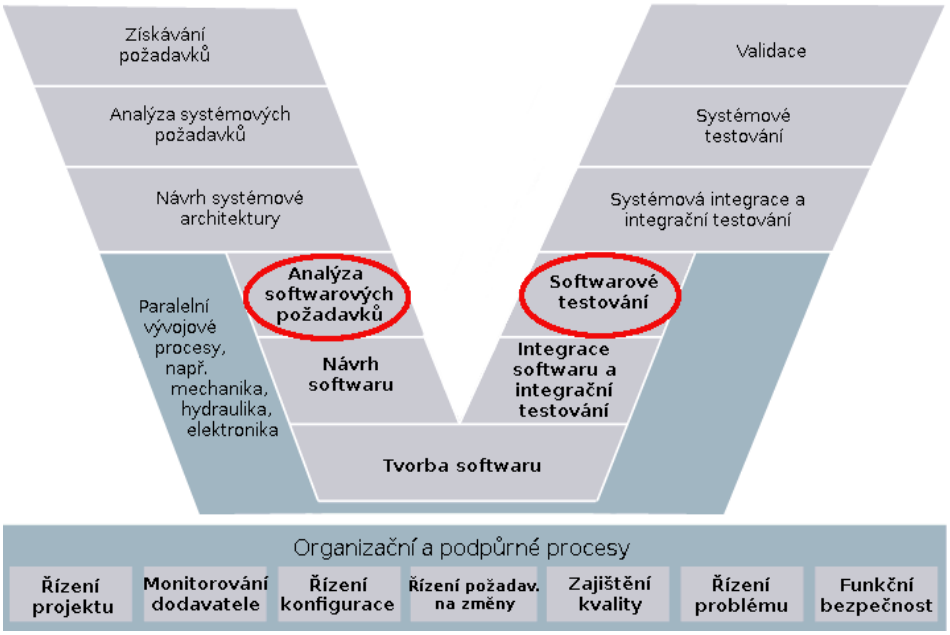
Na obr. 4.2 je zobrazena projektová struktura – kromě vykonávaných procesů na daném pracovišti jsou zakresleny vstupy a výstupy, které si spolupracující partneři poskytují. Z obrázku je dobře patrná vzájemná komunikace mezi nimi. Dále je struktura rozšířena o projektové řízení a zajištění kvality. ZF FRD DTEC funguje především jako komunikační kanál mezi OEM, LS a EDS a poskytuje vstup pro ZF PLZ EDS v podobě projektových softwarových požadavků, které se pak dále zpracovávají. Analýza softwarových požadavků, návrh softwaru a tvorba softwaru

jsou procesy, které se paralelně vykonávají v ZF PLZ EDS a ZF LS. Rozdíl mezi nimi spočívá v typu zaměření. Zatímco ZF LS vyvíjí obecný software, který je společný a přístupný pro všechny projekty na stejné platformě, ZF PLZ EDS se specializuje na zpracování softwaru orientovaného projektově.




*Obr. 4.2: Projektová struktura*

S ohledem na rozsah diplomové práce jsou vybrány dva související procesy V-modelu prováděné v ZF PLZ EDS a ZF PLZ EDT. První proces je „Analýza softwarových požadavků“ z levé vývojové větve a druhý proces je „Softwarové testování“ z pravé testovací větve V-modelu, viz obr. 4.3. Procesy vychází ze standardního procesu popsaného v kapitole 2. Prvky procesního řízení – např. určení vstupů, výstupů je popsáno v kapitole 2.4. a identifikaci jednotlivých rolí lze nalézt v kapitole 2.12.

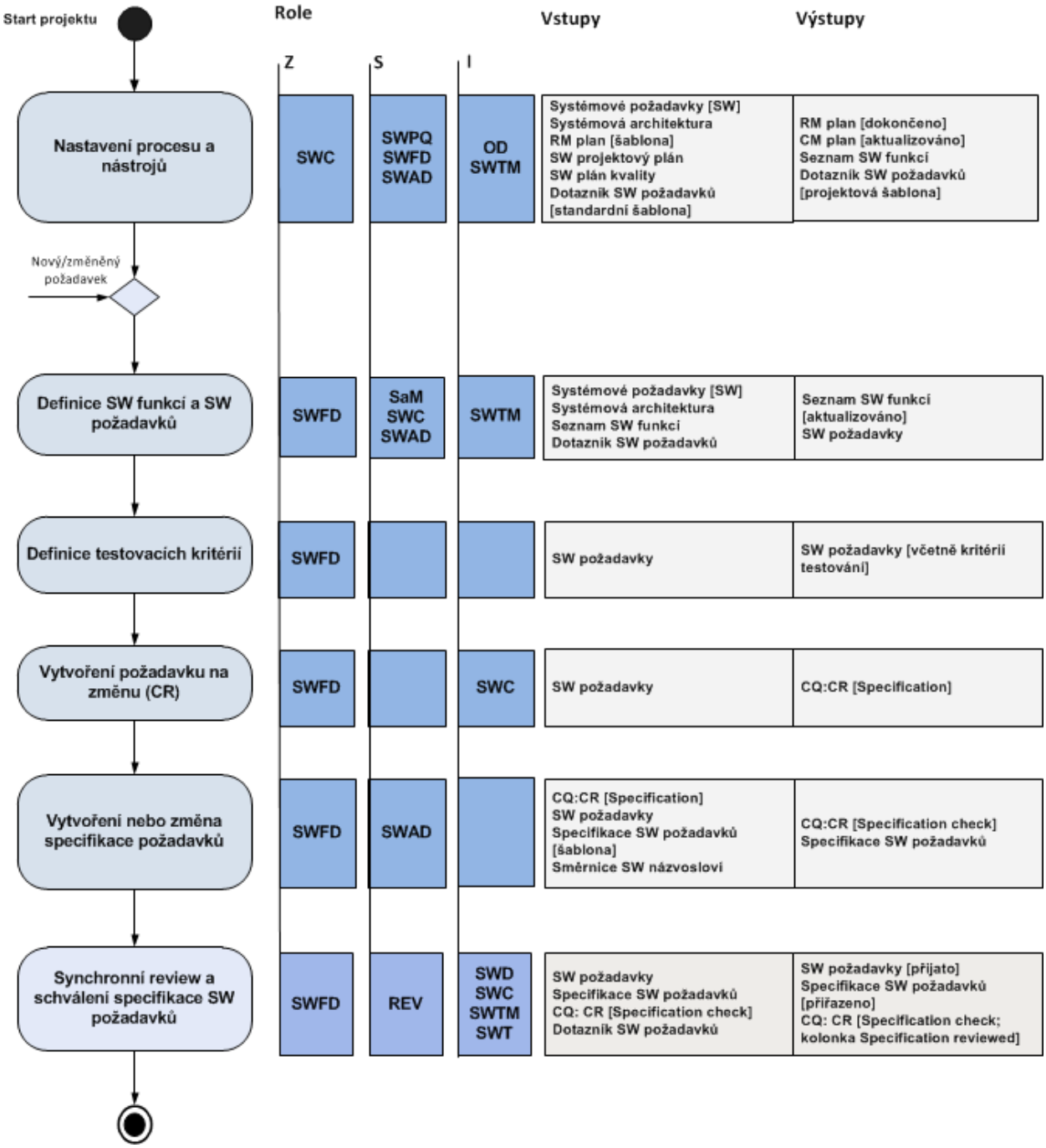



*Obr. 4.3: Vybrané procesy ve V-modelu*

Příklad implementovaných procesů je ilustrován na konkrétním požadavku ve funkci „Lane Departure Warning“. Jedná se o funkci, která má varovat řidiče před nehodou pomocí vibrací do volantu, pokud dotyčný ztratí pozornost nebo usne za jízdy v automobilu na dálnici nebo rychlostní silnici a vozidlo začne svévolně opouštět jízdní pruh.

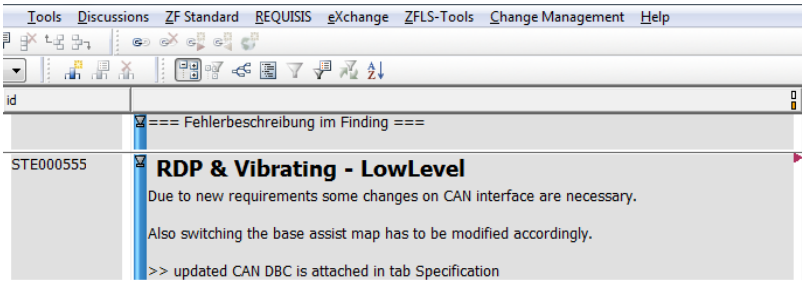
## 4.1 *Analýza softwarových požadavků*

Vývojový diagram procesu „Analýza softwarových požadavků“ je zakreslen na obr. 4.4. Prvním pracovním krokem ve vývojovém diagramu je „Nastavení procesu a nástrojů“. Proces je přizpůsoben podmínkám projektu a jsou připraveny nástroje a databáze, které se budou během realizace projektu používat. Tento krok se na rozdíl od ostatních provádí pouze při zahájení projektu. Mezi jeho výstupy se řadí dokument „plán řízení požadavků“, který definuje, jak se má pracovat s požadavky na software zpravidla za použití nástroje DOORS.



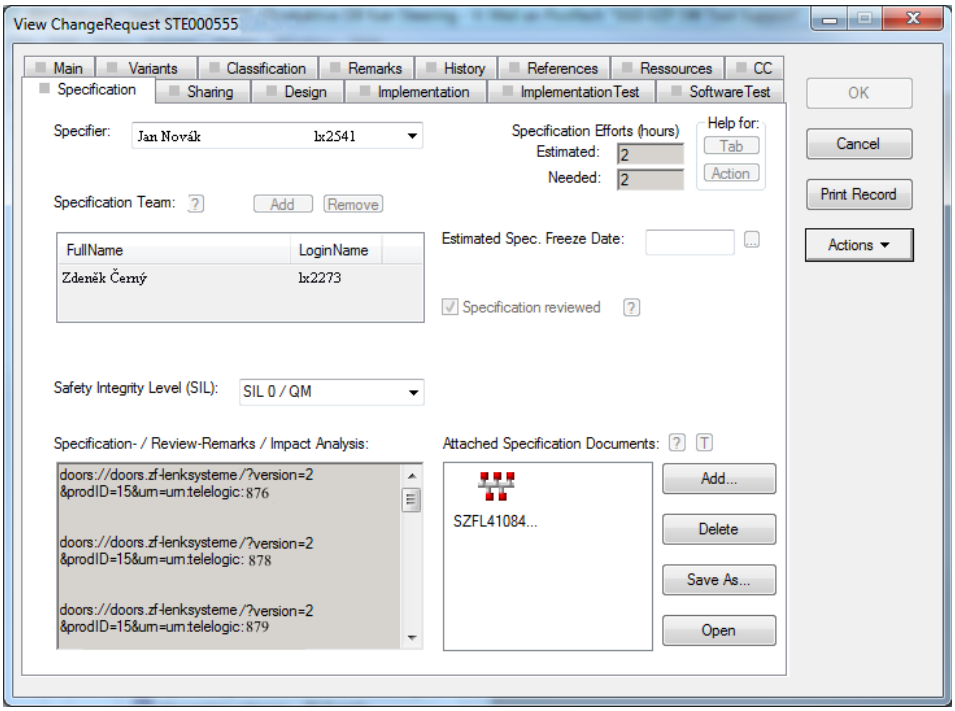

*Obr. 4.4: Vývojový diagram procesu „Analýza softwarových požadavků"*

Pracovní krok „Definice SW funkcí a SW požadavků“ probíhá na základě systémových požadavků a poté je jejich znění zadáno do nástroje DOORS. Konkrétní požadavek – aktualizovat rozhraní CAN, lze vidět na obr. 4.5. Vzhledem k tomu, že se jedná o funkční požadavek, z něhož explicitně vyplývá, co má být testováno, není potřeba do DOORS zadávat testovací kritérium. Pokud by byl požadavek označen jako nefunkční, je nezbytné u něj testovací kritérium určit.




*Obr. 4.5: SW požadavek v nástroji DOORS*

Pracovní krok „Vytvoření požadavku na změnu (CR)“ je uskutečňováno v nástroji ClearQuest. Tento nástroj předepisuje průběh změny a je sledováno, v jaké fázi se změna momentálně nachází, viz obr. 4.6. K jednotlivým činnostem v kartě požadavku na změnu jsou přiřazena jména zodpovědných pracovníků. Dále je zaznamenán odhadovaný čas k plnění úkolu a po dokončení úkolu čas skutečně vynaložený. Důležité je určení, zda jsou na požadavek kladeny nároky z hlediska bezpečnosti, např. kvůli stupni kontrol pracovního výsledku. V uvedeném příkladě je úroveň bezpečnosti klasifikována jako nulová. Požadavek na změnu tedy nemá vliv na způsobení případné nebezpečné události, která by vedla k materiálním škodám, poškození zdraví či ztrátě života.

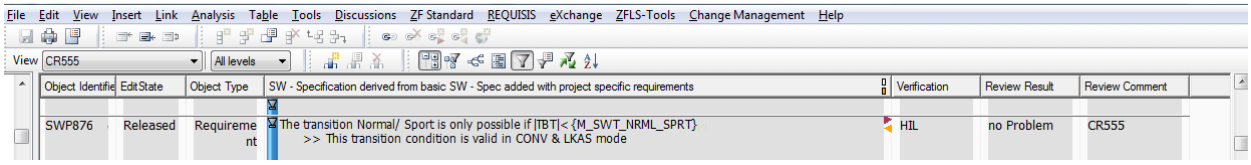



*Obr. 4.6: Požadavek na změnu v nástroji ClearQuest*

Provázanost a dohledatelnost mezi požadavkem na software v nástroji DOORS a požadavkem na změnu v nástroji ClearQuest je uskutečňována pomocí identifikačního čísla (ID). Do nástroje DOORS je prostřednictvím ClearQuest importu nahráno ID požadavku na

změnu a v nástroji ClearQuest je vložen odkaz v políčku „Specification-Remarks“ na daný požadavek v DOORS, viz obr. 4.6.

Dalším pracovním krokem je „Vytvoření nebo změna specifikace požadavků“. Na základě požadavku je v DOORS napsána také jeho specifikace, viz obr. 4.7, kdy je stanovena přesná podmínka chování - přechod z normálního režimu do sportovního je možný, pokud absolutní hodnota z TBT je menší než hodnota konstanty.

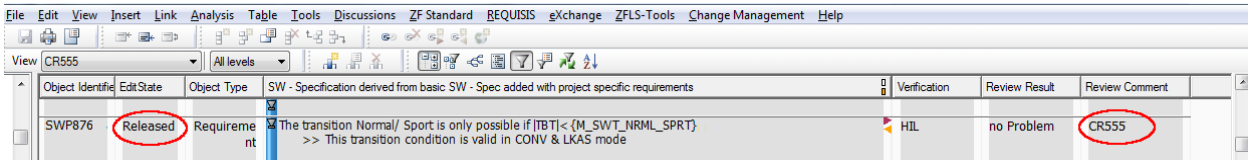



*Obr. 4.7: Specifikace SW požadavku*

Dalším pracovním krokem je „Synchronní review a schválení specifikace SW požadavků“. Specifikaci kontroluje nezávislá osoba s příslušnými kompetencemi. Pokud není nalezena žádná chyba, lze v nástroji ClearQuest zaškrtnout políčko „Specification reviewed“. Schválení specifikace se projeví i v nástroji DOORS, kdy musí být změněn stav z „New“ nebo „Changed“ na „Released“.

Pravidelně každé tři měsíce probíhá v projektu pod záštitou zajištění kvality procesní review, díky kterému je sledováno, na kolik procent jsou všechny procesy vykonávány podle stanovených postupů. V případě, že daný proces nesplňuje definované cíle, jsou určeny a sledovány úkoly k vyřešení nedostatků v řízení projektu.

U procesu „Analýza softwarových požadavků“ byla navržena kritéria, podle kterých je možno sledovat jeho výkonnost. Prvním identifikátorem je žádoucí stav „Released“ u všech požadavků v nástroji DOORS. Dále se výkonnost procesu sleduje kontrolováním, zda jsou všechny požadavky identifikovány, tzn. je zajištěna provázanost identifikačním číslem v nástrojích ClearQuest a DOORS. Identifikace se kontroluje manuálně použitím filtrů. Na obr. 4.8 je znázorněna kontrola stavu „Released“ a identifikačního čísla CR.




*Obr. 4.8: Kontrola kritérií v nástroji DOORS*

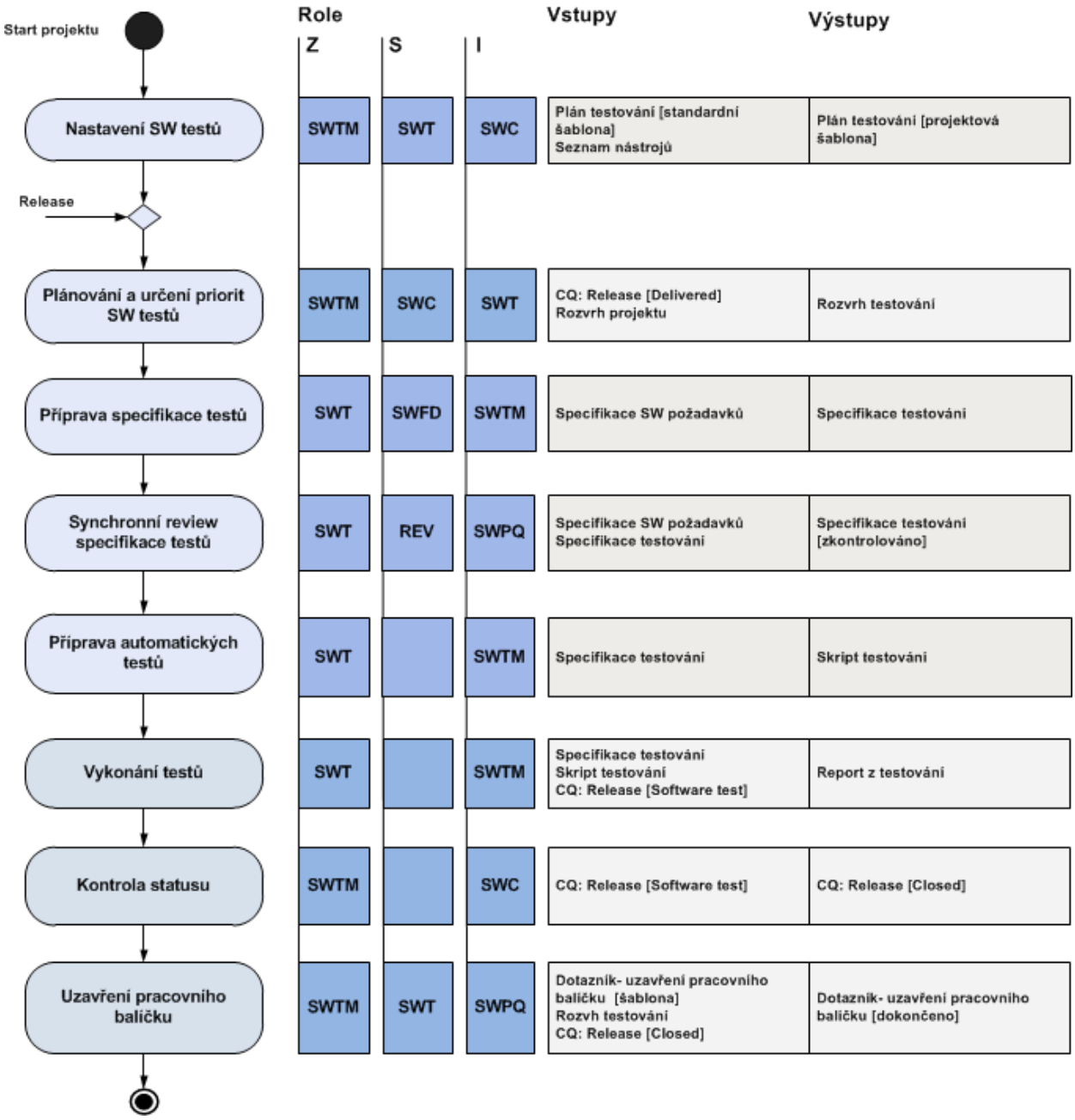
*Tab. 2: Sledování výkonnosti procesu „Analýza softwarových požadavků“*

| **Předmět** | **Zaměření** | **Metoda** | **Cíl kvality / Frekvence** |
|---|---|---|---|
| Požadavky | Stav požadavku | Review: WT | **CK:** Každý SW požadavek je ve stavu „Released“<br><br>**F:** pro každou baseline |
| | Kontrola CR | Review: WT | **CK:** Každý SW požadavek má přiřazen CR<br><br>**F:** pro každou baseline |

*Baseline* oficiálně zachycuje stav vývoje softwaru všech komponent. Obvykle jsou generovány pro důležité milníky v projektovém plánování, např. prototypová výroba, a je oficiálním podkladem pro další vývoj. Změny, které souvisí s danou *baseline* jsou brány jako základ.

## 4.2 *Softwarové testování*

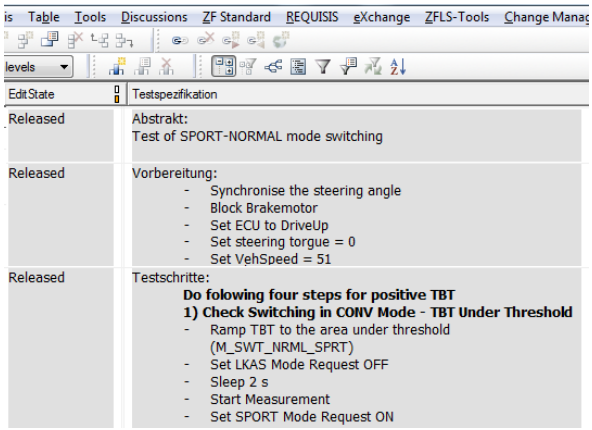
Posloupnost všech činností v procesu “Softwarové testování” je zobrazeno na obr. 4.9. Na začátku projektu se provádí pracovní krok „Nastavení SW testů“, ve kterém jsou nastaveny potřebné nástroje, zajištěny licence a je zvolena projektová šablona dokumentu „plán testování“. V procesu je testován požadavek z předchozí kapitoly.




*Obr. 4.9: Vývojový diagram procesu „Softwarové testování"*

V dalším pracovním kroku se plánuje termín, do kterého musí být testy hotovy. Softwarový koordinátor zadá testerům požadavek, který musí do stanoveného data otestovat. Plánování je aplikováno v nástroji Microsoft Project.

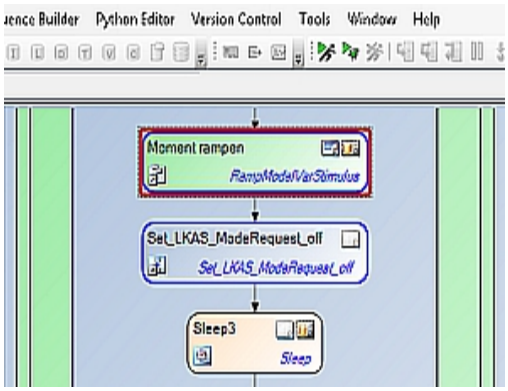
Pracovní krok „Příprava specifikace testů“ je vykonáván, jakmile má tester přiřazen požadavek. V nástroji DOORS začne vytvářet specifikaci testování na základě specifikace požadavků, viz obr. 4.10.




*Obr. 4.10: Specifikace testování*

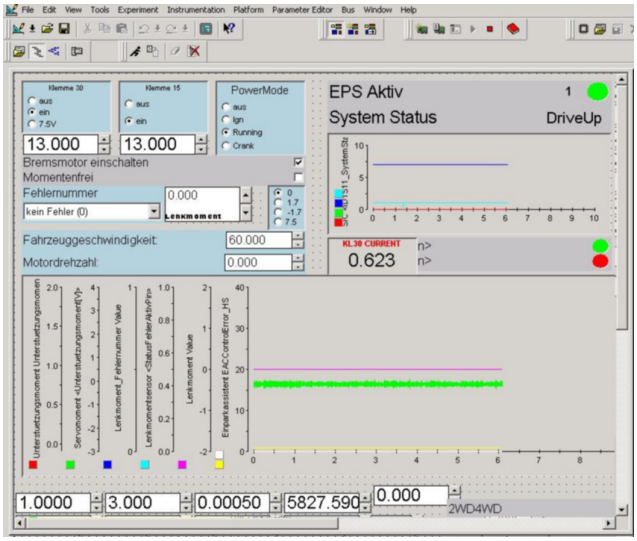
V následujícím pracovním kroku je specifikace testování kontrolována nezávislou osobou. Pokud nejsou nalezeny chyby, lze v nástroji DOORS změnit stav na „Released“.

Pracovní krok „Příprava automatických testů“ spočívá ve vytvoření testovacího scénáře v nástroji AutomationDesk dle specifikace testování, viz obr. 4.11.




*Obr. 4.11: Ukázka testovacího scénáře v AutomationDesk*

Samotné testování probíhá v pracovním kroku „Vykonávání testů“ jednak spuštěním testovacího scénáře z předchozího pracovního kroku a dále ručním zadáváním hodnot a pozorováním chování v nástroji ControlDesk, viz obr. 4.12. V CQ: Release je nastaven stav na „Software test“.

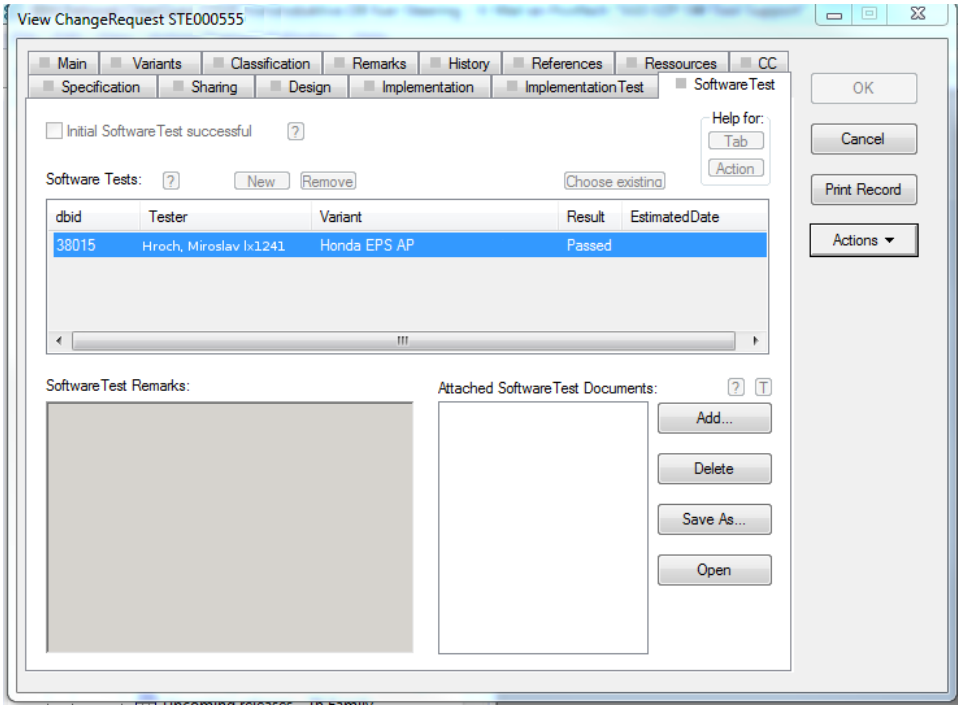



*Obr. 4.12: Příklad testování v nástroji ControlDesk*

V pracovním kroku „Kontrola statusu“ následuje kontrola statusu testování, to znamená, že se zkontroluje, jestli byly všechny testy provedeny. Pokud ano, lze změnit stav v CQ: Release ze „Software test“ na „Closed“.

Posledním krokem je „Uzavření pracovního balíčku“, kdy je prostřednictvím seznamu kontrolních otázek ověřováno, zda bylo vše správně zdokumentováno, archivováno atd.

V procesu softwarové testování se klade důraz na opakovatelnost testů a jejich ukončení v plánovaném termínu. Podmínky při testování musí být vždy zaznamenány, aby mohl být test kdykoliv reprodukovatelný. Podmínky a výsledky z testování jsou obsaženy v dokumentu „report z testování“. Požadavek z kapitoly 4.1 byl otestován s výsledkem „Passed“. Tento výsledek je zaznamenán i v požadavku na změnu v nástroji ClearQuest, viz obr. 4.13. Ukončení testů do stanoveného data je sledováno v nástroji Microsoft Project.




*Obr. 4.13: Výsledek SW testování v nástroji ClearQuest*

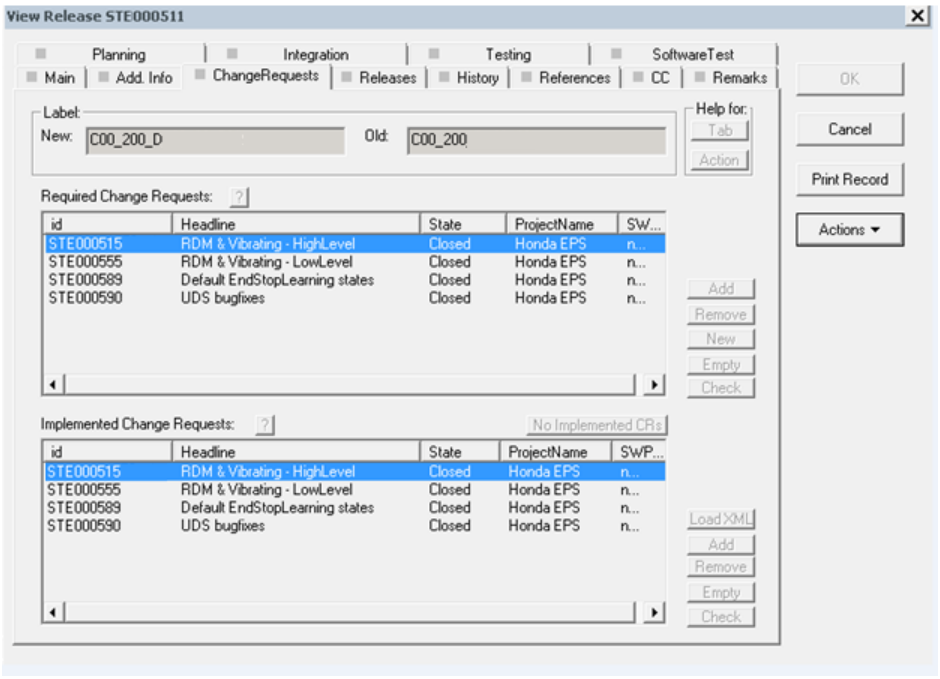
*Tab. 3: Sledování výkonnosti procesu „Softwarové testování"*

| **Předmět** | **Zaměření** | **Metoda** | **Cíl kvality / Frekvence** |
|---|---|---|---|
| SW testování | Test SW požadavků | Metrika v CQ | **CK:** všechny CR jsou ve stavu "Closed"<br><br>**F:** pro každý release |
| | Kontrola příčin chyb v SW po SW testování | Analýza reportů testování | **CK:** 0 chyb ve specifikaci testování<br><br>**F:** měsíčně |
| | Chyby SW po validaci zákazníkem | Sledování spokojenosti zákazníka | **CK:** 0 chyb v dodaném SW<br><br>**F:** čtvrtletně |

SW inženýr kvality vytváří pravidelné reporty stavu kvality projektu, kde jsou zohledněny všechny procesy implementované v projektu. Z pravidelných reportů lze vytvářet statistiky za konkrétní období a sledovat trendy. Případné negativní trendy se řeší nápravnými opatřeními. Pro sledování všech stanovených úkolů na projektu se používá dokument

„projektový seznam úkolů“, ve kterém jsou přiřazeni zodpovědní pracovníci a termíny předpokládaného splnění, a lze v něm sledovat stav daného opatření.

Jedním z výše navrhovaných způsobů, jak se měří výkonnost procesu, je sledování, zda jsou všechny požadavky na změnu v žádoucím stavu. To je zjišťováno v nástroji ClearQuest. Po vybrání požadovaného „Release“ je možno zobrazit všechny CR a shlédnout jejich stav, viz obr. 4.14.

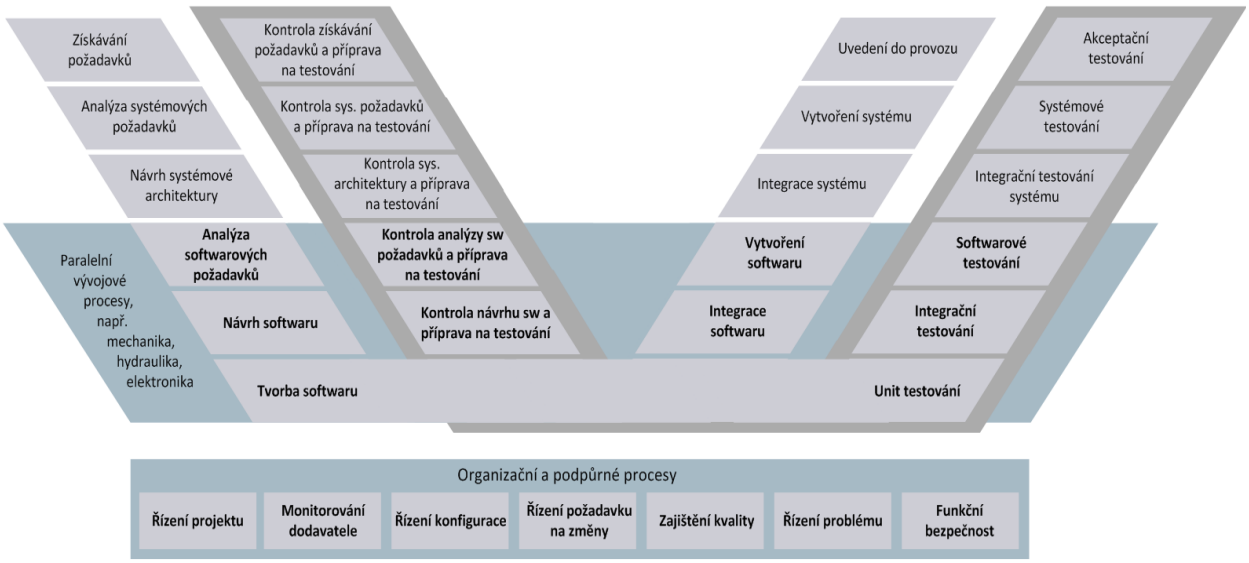



*Obr. 4.14: Release v nástroji ClearQuest*

# 5 Simulace ZF W-modelu

Tato kapitola se zaměřuje na transformaci ZF V-modelu do Herzlichova W-modelu a je zkoumáno, zda použití nového procesního modelu povede k optimalizaci a vyšší efektivitě vývoje softwaru. Okrajově je zváženo ekonomické hledisko.

Jednotlivé navrhované procesy implementované do W-modelu jsou zobrazeny na obr. 5.1. Původní vstupní a výstupní dokumenty ze ZF V-modelu jsou použity i ve W-modelu a pracovní kroky jsou rozděleny podle potřeby. Všechny organizační a podpůrné procesy z dolní části původního procesního modelu jsou zachovány. První „V“ ve W-modelu reprezentuje vývoj softwaru a druhé stínové „V“ jeho testování. V levé části stínového „V“ je testováním myšlena především kontrola vývojového procesu a příprava na budoucí testy.




*Obr. 5.1: Navrhovaný W-model*

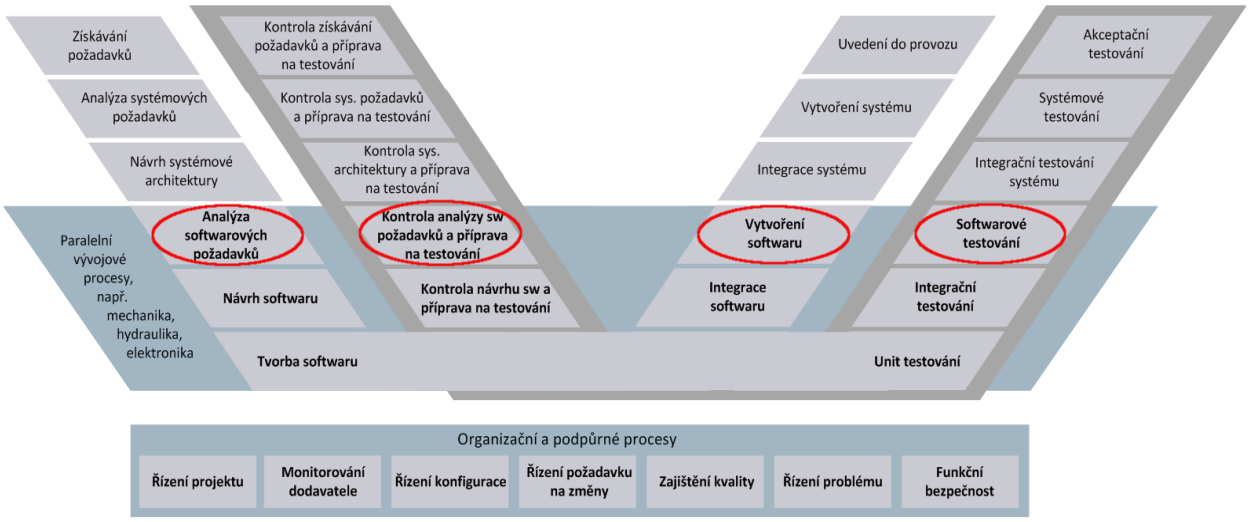
Posloupnost provádění procesů začíná ve vývojové části, poté pokračuje v testovací části a následně přechází do další úrovně vývoje, viz obr. 5.2.


*Obr. 5.2: Pořadí procesů ve W-modelu*

Realizace v ZF V-modelu je zahájena po dokončení procesu předchozího a teprve po implementaci kódu se začíná s přípravou na testy a samotným testováním. ZF W-model se liší

v procesech týkajících se kontroly vývojových procesů a přípravy na testování. Po dokončení všech pracovních kroků souvisejících s kontrolou předchozího vývojového procesu, lze současně vykonávat fázi přípravy testování s vývojovým procesem na další úrovni. Konkrétně u procesu „Kontrola analýzy softwarových požadavků a příprava na testování“ je možnost zpracovávat druhou fázi tohoto procesu současně s následujícím vývojovým procesem „Návrh softwaru“. Paralelní vykonávání obou procesů lze provádět ze dvou důvodů. Prvním je ukončené schválení dokumentu „softwarová specifikace požadavků“, který je základním vstupem do vývojového procesu. Druhým důvodem je skutečnost, že dokument „specifikace testování“ se stává vstupem až v pozdějším procesu „Softwarové testování“.

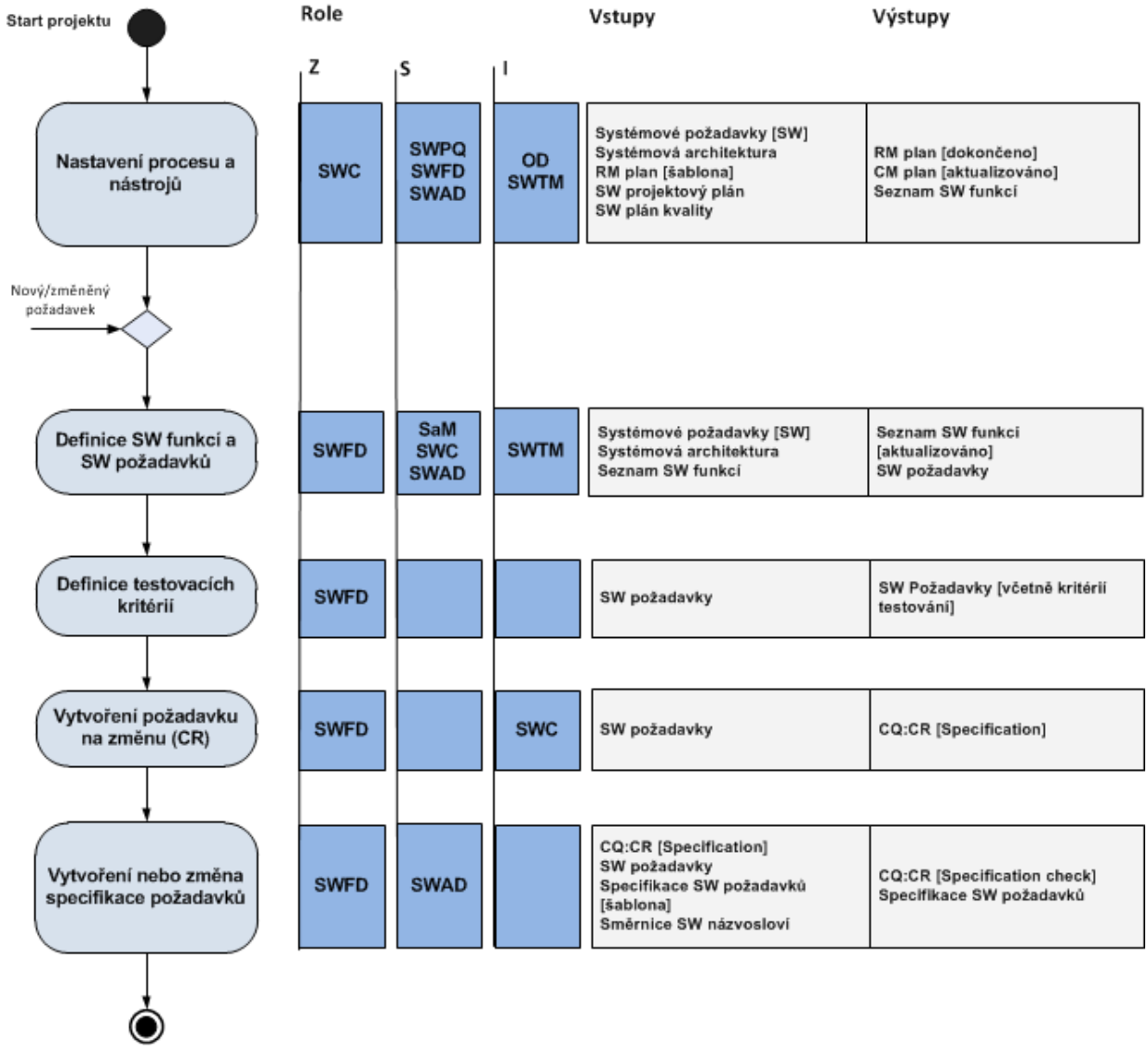
V ZF W-modelu jsou podrobně zpracovány procesy „Analýza softwarových požadavků“, „Kontrola analýzy softwarových požadavků a příprava na testování“, „Vytvoření softwaru“ a „Softwarové testování“, viz obr. 5.3. U každého z nich je navrhnut vývojový diagram. Jako podklad pro tyto nově navrhované procesy byly použity výchozí procesy „Analýza softwarových požadavků“ a „Softwarové testování“ ze ZF V-modelu popsané v kapitole 4.




*Obr. 5.3: Vybrané procesy k analýze*

První vykonávaný proces „Analýza softwarových požadavků“ v ZF W-modelu u podsystému software je nazván totožně jako v ZF V-modelu. Vývojový diagram je ve srovnání s původním upraven hlavně vyjmutím posledního pracovního kroku spojeného s kontrolní činností, který je použit v 1. fázi druhého procesu. Schéma je zobrazeno na obr. 5.4. Podstata procesu spočívá především v identifikaci softwarových funkcí a požadavků, vytvoření testovacích kritérií u nefunkčních požadavků a zhotovení specifikace požadavků. Za správné provedení všech pracovních kroků je zodpovědný funkční vývojář. Výjimku tvoří pouze

jednorázově prováděný pracovní krok softwarovým koordinátorem při zahájení projektu „Nastavení procesu a nástrojů“.

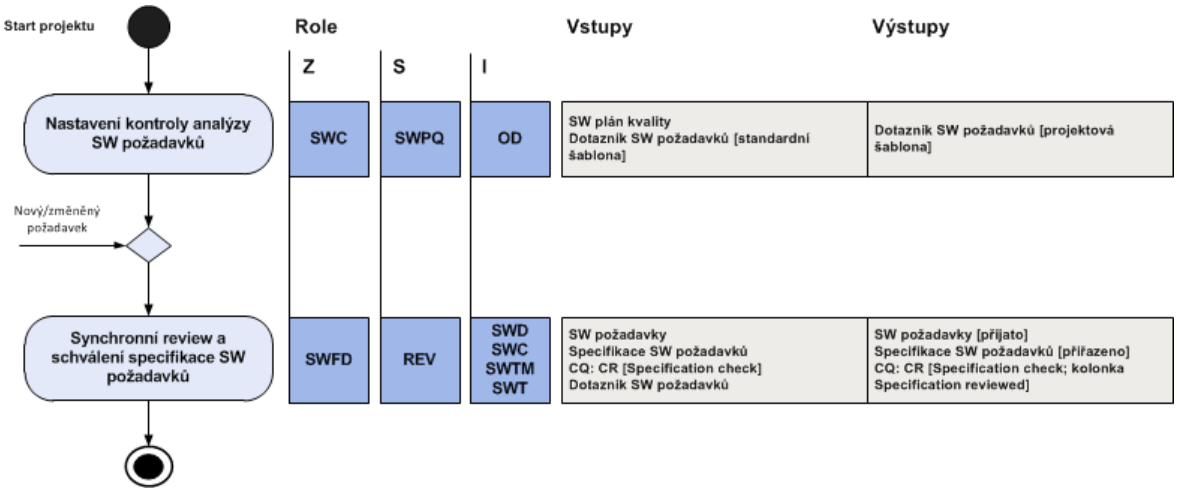



*Obr. 5.4: Vývojový diagram procesu „Analýza softwarových požadavků“ ve W-modelu*

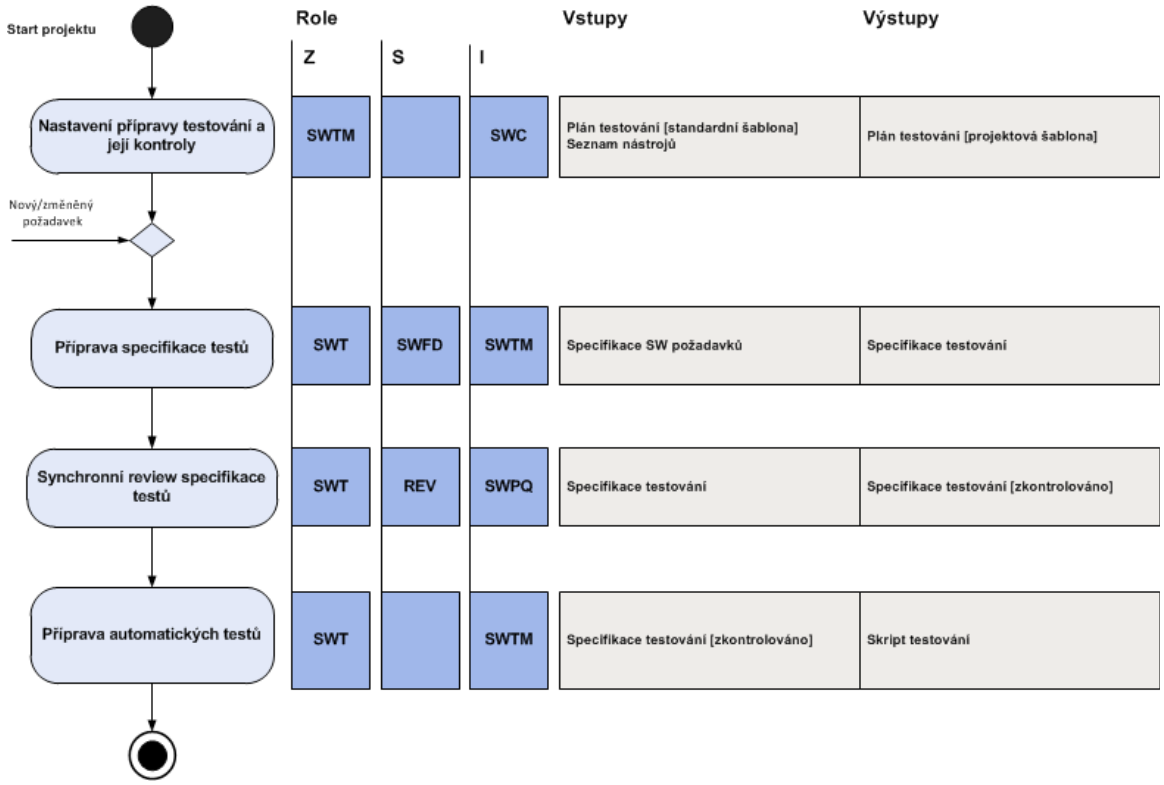
Proces „Kontrola analýzy softwarových požadavků a příprava na testování“ lze rozdělit do dvou fází, viz obr. 5.5 a obr. 5.6. První fáze se specializuje pouze na kontrolu předchozího procesu a reviduje, zda splňuje softwarová specifikace požadavků všechna kritéria a neobsahuje chyby. Nově byl vytvořen pracovní krok „Nastavení kontroly analýzy SW požadavků“ kvůli správnému vytvoření projektového dotazníku SW požadavků, který slouží jako podklad ke kontrole specifikace požadavků. Ve V-modelu je tento pracovní krok součástí pracovního kroku „Nastavení procesu a nástrojů“. Druhá část vývojového diagramu je zaměřena na přípravu testování, tzn. tvorbu stěžejního dokumentu „specifikace testování“ a následnou kontrolu, jestli je dokument bez chyb. Pro navržení tohoto procesu byly vybrány všechny vhodné pracovní

kroky z vývojového diagramu „Softwarové testování“ v ZF V-modelu.

Zatímco v první fázi procesu vykonává pracovní úkony softwarový koordinátor a softwarový funkční vývojář, ve druhé fázi je za plnění pracovních kroků odpovědný tester a manažer testerů.

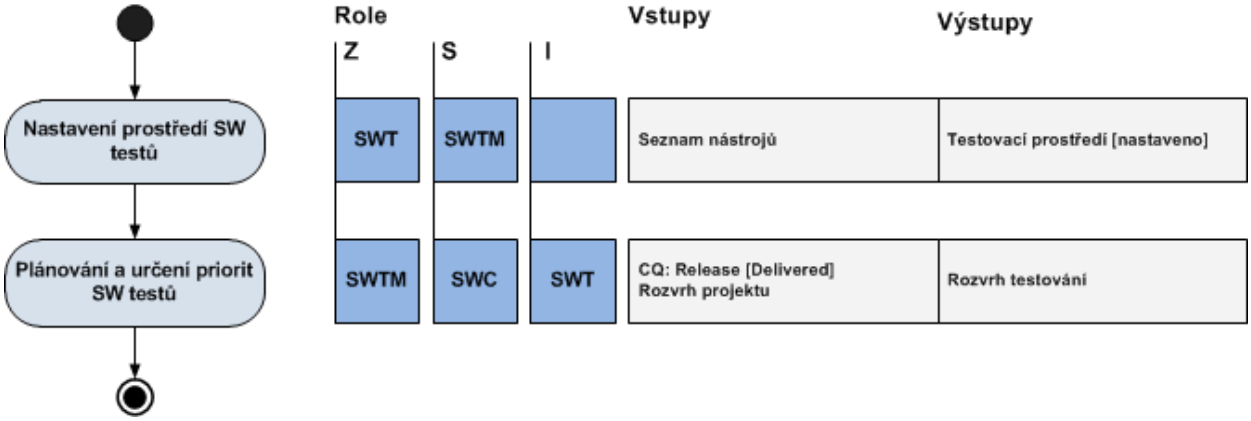



*Obr. 5.5: Vývojový diagram procesu „Kontrola analýzy softwarových požadavků a příprava na testování“ -1.část*



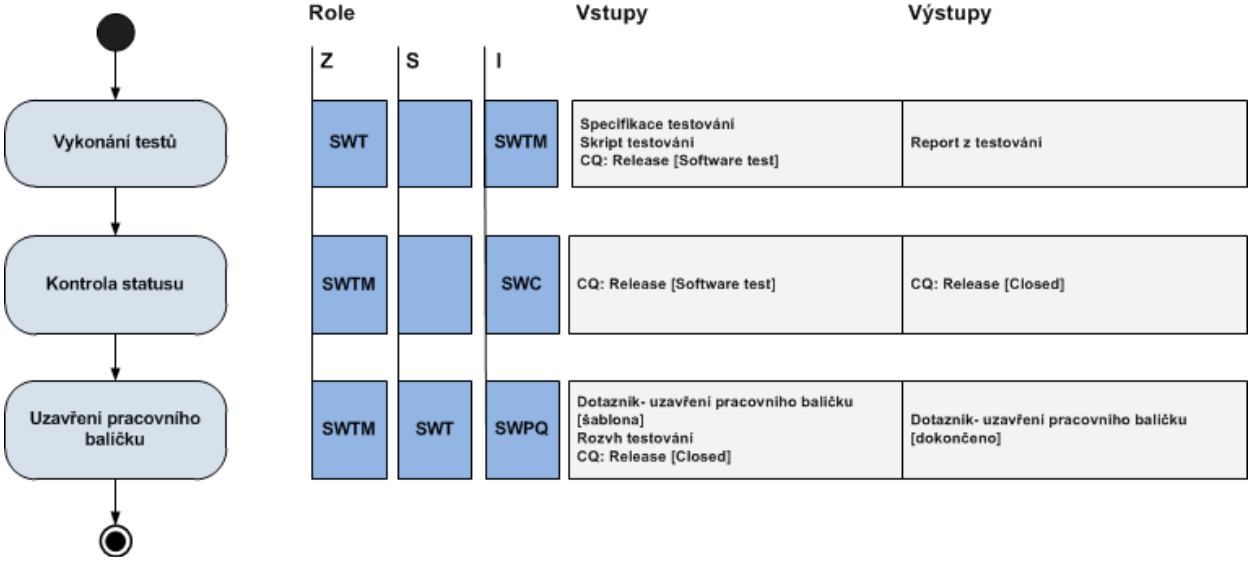

*Obr. 5.6: Vývojový diagram procesu „Kontrola analýzy softwarových požadavků a příprava na testování“ -2.část*

Cílem procesu „Vytvoření softwaru“ je nastavení vhodného prostředí pro provádění softwarového testování a naplánování jednotlivých testů dle přiřazených priorit, viz obr. 5.7. Pracovní krok „Plánování a určení priorit SW testů“ je převzat z procesu „Softwarové testování“ v ZF V-modelu a pracovní krok „Nastavení prostředí SW testů“ je vytvořen nově. Vstupy a výstupy jsou vhodně zvoleny.




*Obr. 5.7: Vývojový diagram procesu „Vytvoření softwaru“ ve W-modelu*

Proces „Softwarové testování“ se zabývá samotným testováním. Schéma vývojového diagramu je navrženo na obr.5.8.




*Obr. 5.8: Vývojový diagram procesu „Softwarové testování" ve W-modelu*

## 5.1 *Ekonomické vyhodnocení modelů*

Určení, který z modelů je na provádění procesů optimálnější, je vyhodnoceno pomocí stanovení nákladů na jednotlivé pracovní kroky při zpracování požadavku ze 4. a 5. kapitoly. V níže uvedených tabulkách jsou rozúčtované náklady na činnosti podle jednoduchého vzorce:

celkové náklady = čas strávený aktivitou × hodinová průměrná mzda příslušného pracovníka.

Průměrná mzda na danou pozici byla odhadnuta na základě dostupných informací na internetových stránkách www.nabidky-prace.cz. Čas u pracovních kroků byl stanoven ze skutečných odhadů. Přesto je nutné brát určené časy pouze orientačně, protože každý pracovník má jinou úroveň zkušeností, každému může trvat splnění úkolu odlišně a hlavně každý požadavek na změnu je jinak náročný.

Vyúčtování pracovních kroků je ukázáno na procesech „Analýza softwarových požadavků“ a „Softwarové testování“ ze ZF V-modelu. Ze ZF W-modelu jsou pro porovnání vyúčtovány procesy simulované v kapitole 5 .

*Tab. 4: Průměrná hodinová mzda pracovníků*

| Role | Prům. hod. mzda [Kč] |
|---|---|
| Tester softwaru – SWT | 156 |
| Manažer testování softwaru – SWTM | 250 |
| Softwarový koordinátor – SWC | 250 |
| Softwarový funkční vývojář – SWFD | 156 |
| Softwarový projektový manažer kvality – SWPQ | 220 |

*Tab. 5: Finanční náklady na proces „Analýza softwarových požadavků“ ve V-modelu*

| Pracovní krok | Odpovědnost | Čas [hod] | Celkem [Kč] |
|---|---|---|---|
| Nastavení procesu a nástrojů | SWC | 80 | 20000 |
| Definice SW funkcí a SW požadavků | SWFD | 4 | 624 |
| Definice testovacích kritérií | SWFD | 2 | 312 |
| Vytvoření požadavku na změnu (CR) | SWFD | 0,5 | 78 |
| Vytvoření nebo změna specifikace požadavků | SWFD | 2 | 312 |
| Synchronní review a schválení specifikace SW požadavků | SWFD | 1 | 156 |
| **SOUČET** | | **89,5** | **21482** |

*Tab. 6:Finanční náklady na proces „Softwarové testování“ ve V-modelu*

| Pracovní krok | Odpovědnost | Čas [hod] | Celkem [Kč] |
|---|---|---|---|
| Nastavení SW testů | SWC | 80 | 20000 |
| Plánování a určení priorit SW testů | SWC | 1 | 250 |
| Příprava specifikace testů | SWT | 2 | 312 |
| Synchronní review specifikace testů | SWTM | 1 | 250 |
| Příprava automatických testů | SWT | 2 | 312 |
| Vykonávání testů | SWT | 3 | 468 |
| Kontrola statusu | SWTM | 2 | 500 |
| Uzavření pracovního balíčku | SWTM | 1 | 250 |
| **SOUČET** | | **92** | **22342** |

*Tab. 7:Finanční náklady na proces „Analýza softwarových požadavků“ ve W-modelu*

| Pracovní krok | Odpovědnost | Čas [hod] | Celkem [Kč] |
|---|---|---|---|
| Nastavení procesu a nástrojů | SWC | 72 | 18000 |
| Definice SW funkcí a SW požadavků | SWFD | 4 | 624 |
| Definice testovacích kritérií | SWFD | 2 | 312 |
| Vytvoření požadavku na změnu (CR) | SWFD | 0,5 | 78 |
| Vytvoření nebo změna specifikace požadavků | SWFD | 2 | 312 |
| **SOUČET** | | **80,5** | **19326** |

*Tab. 8:Finanční náklady na proces „Kontrola analýzy SW požadavků a příprava na testování“ ve W-modelu*

| Pracovní krok | Odpovědnost | Čas [hod] | Celkem [Kč] |
|---|---|---|---|
| Nastavení kontroly analýzy SW požadavků | SWC | 8 | 2000 |
| Synchronní review a schválení specifikace SW požadavků | SWFD | 1 | 156 |
| Nastavení přípravy testování a její kontroly | SWC | 80 | 20000 |
| Příprava specifikace testů | SWT | 2 | 312 |
| Synchronní review specifikace testů | SWT | 0,5 | 78 |
| Příprava automatických testů | SWT | 2 | 312 |
| **SOUČET** | | **93,5** | **22858** |

*Tab. 9:Finanční náklady na proces „Vytvoření softwaru“ ve W-modelu*

| Pracovní krok | Odpovědnost | Čas [hod] | Celkem [Kč] |
|---|---|---|---|
| Nastavení prostředí SW testů | SWT | 0,5 | 78 |
| Plánování a určení priorit SW testů | SWC | 1 | 250 |
| **SOUČET** | | **1,5** | **328** |

*Tab. 10:Finanční náklady na proces „Softwarové testování“ ve W-modelu*

| **Pracovní krok** | **Odpovědnost** | **Čas [hod]** | **Celkem [Kč]** |
|---|---|---|---|
| Vykonání testů | SWT | 3,5 | 546 |
| Kontrola statusu | SWTM | 2 | 500 |
| Uzavření pracovního balíčku | SWTM | 1 | 250 |
| **SOUČET** | | **6,5** | **1296** |

*Tab. 11: Sumarizace celkových nákladů na jednotlivé procesy*

| **TYP MODELU** | V-model | W-model |
|---|---|---|
| **NÁKLADY [Kč]** | 43824 | 43808 |
| **ČAS [hod]** | 181,5 | 182 |

Z výsledků lze konstatovat, že jsou nejdražší jednorázové pracovní kroky při zahájení projektu – nastavování procesů. Důvodem je, že ostatní finance jsou spjaty pouze se zpracováním jednoho požadavku. Po odfiltrování činností prováděných pouze na začátku projektu jsou výsledky mezi modely poměrově stejné. Čas i vynaložené finanční prostředky jsou v obou modelech srovnatelné a drobné odchylky lze zanedbat.

*Tab. 12:Sumarizace celkových nákladů bez procesů prováděných při zahájení projektu*

| **TYP MODELU** | V-model | W-model |
|---|---|---|
| **NÁKLADY [Kč]** | 3824 | 3808 |
| **ČAS [hod]** | 21,5 | 22 |

## 5.2 *Zhodnocení ZF W-modelu*

Je zřejmé, že ZF W-model je graficky více přesný než ZF V-model, avšak posloupnost provádění jednotlivých procesů v ZF W-modelu nemusí být nezasvěcenému uživateli bez detailního objasnění pochopitelná.

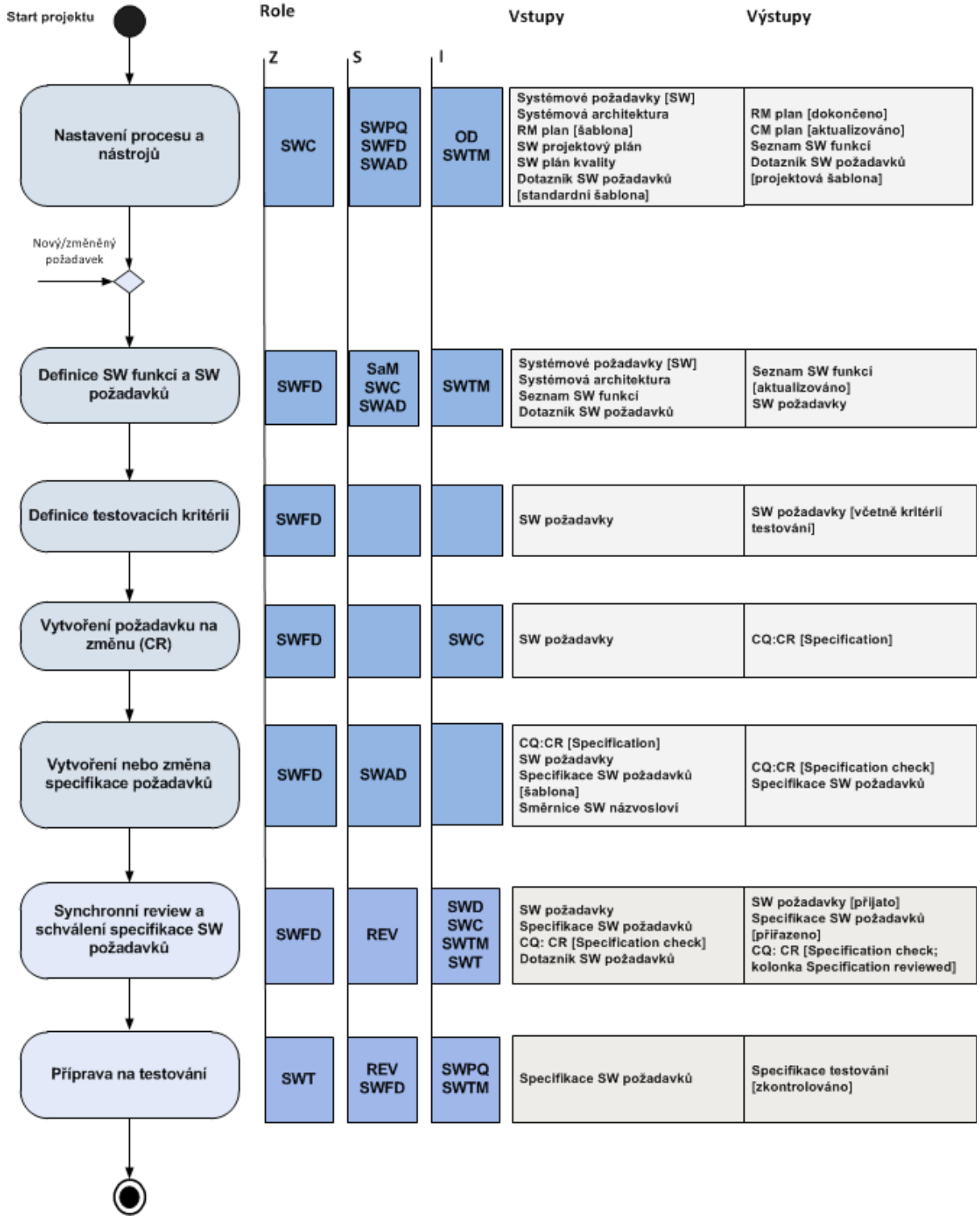
Díky tvorbě „specifikace testování“ jsou testeři zapojeni do projektu od jeho počátku, mohou detekovat případné nedostatky souvisejícího vývojového procesu ve značném předstihu. Jejich zpětná vazba napomůže snížení nákladů díky včasnému odstranění jejich příčiny. Nevýhodou jsou však existující rizika. Dokument je zpracováván v předstihu a používán až v pozdějším procesu. Vykonavatel samotného testování navíc nemusí být autorem specifikace testování. V ideálním stavu se předpokládá, že specifikace bude tak srozumitelná, že podle ní

jakýkoli tester bude schopen provést softwarový test. Je tomu tak ve skutečnosti? Někdy se může stát, že obsah specifikace testování bude pro vykonavatele testů nepřehledný. V tomto případě musí tento pracovník specifikaci vhodně upravit a odstranit nedostatky. Situace může být zapříčiněna např. rozdílnou úrovní zkušeností pracovníků nebo jen špatným provedením práce kvůli lidskému faktoru. Předělávání již schválené specifikace testování vede samozřejmě k časovým ztrátám. Pokud je osoba, která vykonává testy, zároveň i autorem specifikace, je téměř odstraněno riziko spojené s nepochopením obsahu dokumentu. Pravděpodobnost oprav ve specifikaci z důvodu nejasného významu by měla být ovšem nízká, neboť každá zhotovená specifikace prochází kontrolou od pověřené osoby.

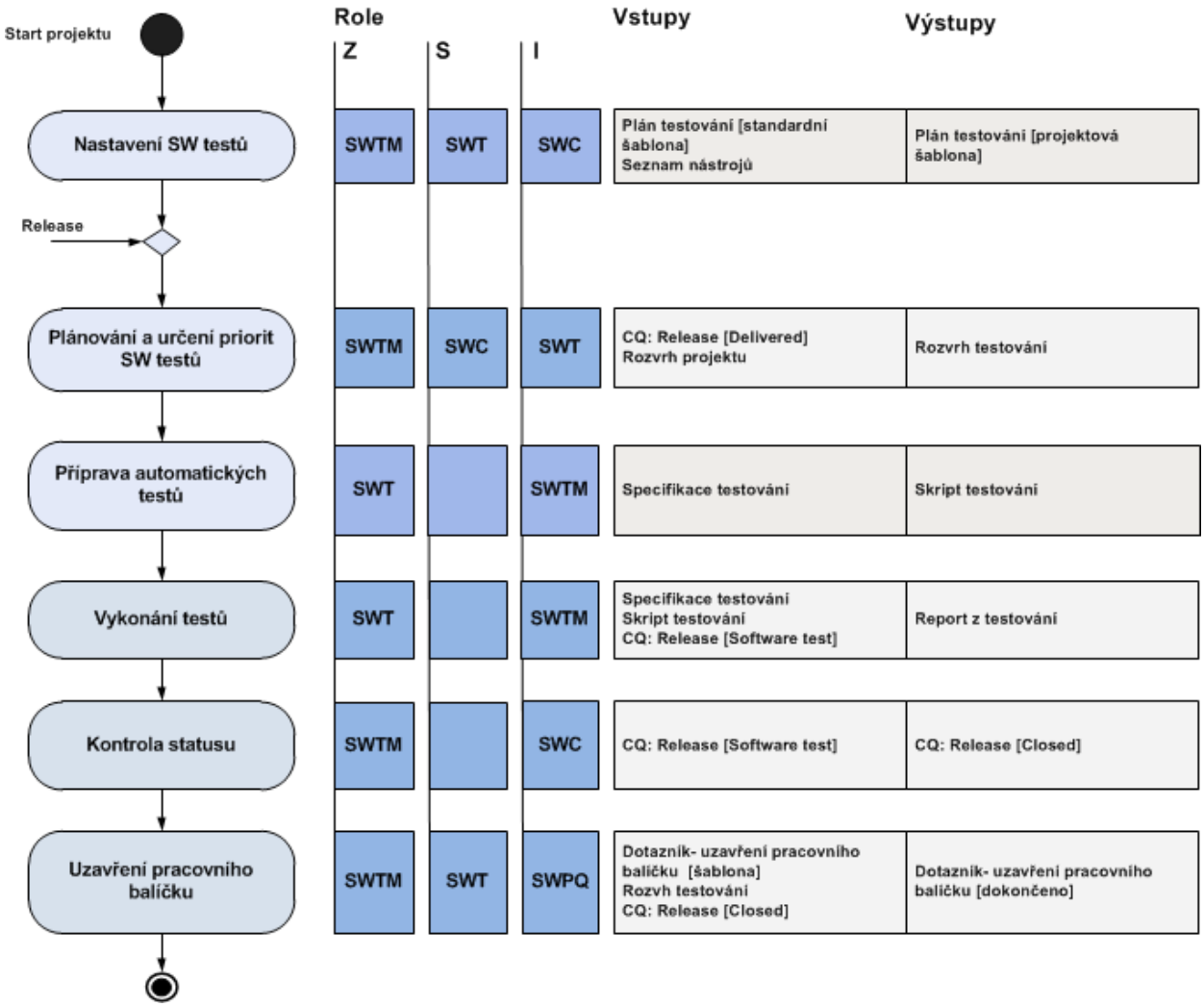
Nezřídka se stává, že zákazník během realizace projektu rozšiřují své požadavky na výsledný produkt. Pokud přibude nový požadavek, který je v souladu s ostatními, softwarový tester specifikaci testování pouze rozšíří. V momentě, kdy je přijat požadavek, který je v kolizi s ostatními, musí softwarový tester celou specifikaci testování přepracovat. Kvůli časnému zhotovení dokumentu a jeho pozdější potřeby se může stát, že si testeři již nebudou detailně pamatovat obsah tohoto dokumentu a budou se v něm muset znovu zorientovat. Opakované obeznámení se specifikací testování zapříčiní jisté časové prodlevy, které jsou z hlediska projektového času zanedbatelné v porovnání s ušetřeným časem během současného provádění dvou procesů („Kontrola analýzy softwarových požadavků a příprava na testování“ a „Návrh softwaru“). Z hlediska celkového času stráveného na projektu, se časový přírůstek samozřejmě projeví, obzvláště v případě zpracovávání častých nových požadavků, kdy je nutná neustálá aktualizace specifikace testování. Omezit příchod nových požadavků od zákazníka lze na základě domluvy, kdy je stanoven nejpozdější možný termín příjmu nových požadavků.

ZF W-model je sice ve srovnání s ZF V-modelem graficky složitější, ale pro znalé osoby se může jevit přesnější. Jsou v něm explicitně znázorněny procesy, které jsou v ZF V-modelu implementovány až ve vývojových diagramech. Oproti ZF W-modelu postrádá ZF V-model výhodu v podobě dřívějšího sepsání specifikace testování. Toho lze ale v ZF V-modelu docílit přidáním pracovního kroku „Příprava na testování" do vývojového diagramu procesu „Analýza softwarových požadavků", viz obr. 5.9. Pracovní krok zahrnuje tvorbu a kontrolu specifikace testování z procesu "Softwarové testování". Po zavedení tohoto zlepšení již nepřichází ZF V-model o žádnou podstatnou výhodu ZF W-modelu a navíc jsou v něm procesy znázorněny jednoduše a přehledně. Pro úplnost navrhované změny pracovních kroků je na obr. 5.10

znázorněn i aktualizovaný vývojový diagram procesu „Softwarové testování“. Doporučením je nezavádět W-model, ale upravit pracovní kroky v současném V-modelu.




*Obr. 5.9: Proces „Analýza softwarových požadavků“ v ZF V-modelu s novým implementovaným pracovním krokem „Příprava na testování“*




*Obr. 5.10: Upravený vývojový diagram procesu „Softwarové testování" v ZF V-modelu*

## 5.3 *Optimalizace procesu pomocí výběru vhodnějšího nástroje*

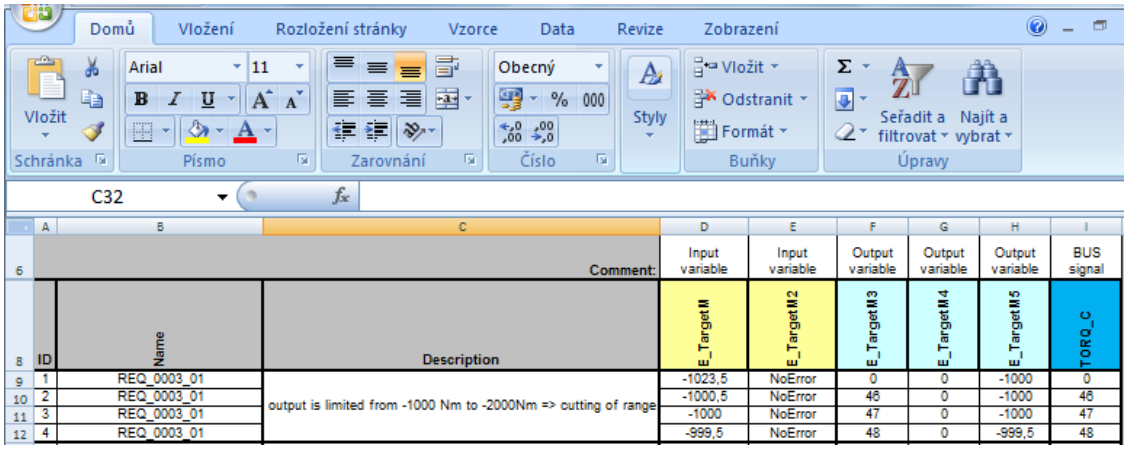
Další možností, jak optimalizovat procesy, je zvolení vhodnějšího nástroje. V procesu „Softwarové testování“ je v některých projektech pro zpracování specifikace testování používán nástroj Excel místo nástroje DOORS. Kapitola se soustředí na porovnání těchto dvou nástrojů.

DOORS (Dynamic Object Oriented Requirement System) je, jak již bylo zmíněno v kapitole 2.4 databázový systém pro správu požadavků. Nástroj umožňuje uchovat, sledovat, analyzovat a řídit požadavky při zachování souladu s předpisy a normami. Centrální správa požadavků podporuje a usnadňuje spolupráci týmu. V celosvětovém měřítku je nástroj v porovnání s nástroji stejného zaměření velmi rozšířen.

Microsoft Excel je součástí balíčku Microsoft Office a má širokou škálu použití, proto je v současnosti po celém světě hojně využíván. Jeho cílem je, aby vyhovoval kancelářskému prostředí. V Excelu je možnost pracovat s programovacím jazykem VBA (Visual Basic for Application), jehož prostřednictvím lze vytvářet nové funkce a tím zajistit upravení souborů na míru podle osobních potřeb. Pro dlouhodobé aplikace ovšem není VBA správnou volbou, neboť ve většině případů se skripty VBA obtížně udržují a aktualizují.

Zpracování specifikace testování v obou nástrojích přináší několik rozdílností. Hodnocení a zaměření na odlišnosti je uvažováno z hlediska způsobu vytváření specifikace testování a její provázaností a dohledatelností s požadavky. Dohledatelnost je důležitá kvůli transparentnosti vývoje softwaru a zajištění kvality.

V nástroji Excel je pro sepsání specifikace testování využívaná jednoduchá šablona, viz obr. 5.11. Používání jednotné šablony přispívá k rychlému zorientování v testovacích případech, což vede k vyšší efektivitě práce testerů. V nástroji DOORS k dispozici žádná šablona není a proto se zápis specifikace může odlišovat v závislosti na autorovi. S tím úzce souvisí odlišný styl psaní specifikace testování. Zatímco v nástroji DOORS je specifikace obecná, viz předchozí obr. 4.10, v nástroji Excel je jasná přehledná tabulka vstupních a výstupních hodnot, viz obr. 5.11. Je tedy zcela zřejmé, co se má zadávat a co očekávat. Tento způsob je ideální pro začátečníky, neboť tabulku pochopí všichni stejně a není zde prostor pro odlišný výklad. Výhodou je i možnost různých výpočtů, tato možnost v nástroji DOORS umožněna není. Způsob psaní specifikace v nástroji Excel není ovšem vhodný pro všechny typy požadavků. Např. pro požadavky, které mají být otestovány cyklicky je vhodnější zápis specifikace v nástroji DOORS, neboť vypracovávání tabulky v nástroji Excel by bylo velmi zdlouhavé a náročné.




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 6 | | | Comment: | Input variable | Input variable | Output variable | Output variable | Output variable | BUS signal |
| 8 | ID | Name | Description | E_TargetM | E_TargetM2 | E_TargetM3 | E_TargetM4 | E_TargetM5 | TORQ_C |
| 9 | 1 | REQ_0003_01 | output is limited from -1000 Nm to -2000Nm => cutting of range | -1023,5 | NoError | 0 | 0 | -1000 | 0 |
| 10 | 2 | REQ_0003_01 | | -1000,5 | NoError | 46 | 0 | -1000 | 46 |
| 11 | 3 | REQ_0003_01 | | -1000 | NoError | 47 | 0 | -1000 | 47 |
| 12 | 4 | REQ_0003_01 | | -999,5 | NoError | 48 | 0 | -999,5 | 48 |

*Obr. 5.11: Ukázka specifikace testování v nástroji Excel*

V nástroji DOORS je nepochybně snadná dohledatelnost požadavků na změnu. Provázanost mezi nástroji ClearQuest a DOORS je vysvětlena v předchozí kapitole. Nevýhoda může být spatřována v prostředí nástroje, které se nejeví jako uživatelsky přátelské. Excel je uživatelsky mnohem přívětivější, ale je zde komplikovanější provázanost mezi specifikací požadavků, požadavky na změnu a specifikací testování. V nástroji Excel se ručně dopisuje identifikační číslo na požadavek na změnu a identifikační číslo daného požadavku. Požadavky jsou uchovávány v dokumentech různých formátů např. doc, pdf. Uvnitř dokumentů musí být ručně dopsáno přiřazené ID požadavku. Název dokumentu s požadavky je nazván stejně jako dokument v nástroji Excel, tím je zajištěna jistá provázanost. V nástroji ClearQuest je zadán název dokumentu s požadavky a identifikační číslo požadavku.

Na začátku projektu v pracovním kroku "Nastavení SW testů" musí být zvážena všechna možná rizika a podle nich by poté měl být zvolen jeden z nástrojů. Zahrnuto by mělo být i ekonomické hledisko. Licence do nástroje DOORS je až 12x dražší než pořizovací cena nástroje Excel. Jedná se ale o síťovou licenci, kterou využívá velký počet uživatelů. Oproti nástroji DOORS patří nástroj Excel ve společnosti do standardního programového vybavení každého počítače.

Obecně však lze říci, že nástroj DOORS umožňuje mnohem sofistikovanější správu požadavků a proto je vhodnější. Podporuje snadnou dohledatelnost a propojení ID mezi softwarovými požadavky, specifikací testování a požadavky na změny. Snadným prolinkováním ID je šetřen čas v porovnání s aplikovaným způsobem v nástroji Excel a je snížena pravděpodobnost chyb. Názor softwarových testerů ze ZF PLZ o tom, který z nástrojů je optimálnější používat, byl odlišný, ale většina z dotázaných by upřednostnila nástroj DOORS.

# Závěr

Předkládaná diplomová práce představila různé typy modelů životních cyklů softwarového vývoje. Důraz je kladen na V-model, který je v praxi využíván i ve společnosti ZF Engineering Plzeň, s.r.o. v oddělení elektroniky. Všechny procesy, které ZF V-model obsahuje, jsou popsány v jednotlivých kapitolách, což přispívá k snadnějšímu pochopení fungování vývoje softwaru podle výše jmenovaného modelu.

Druhá polovina práce se zabývala nejen procesním řízením v projektu Honda, kde byly navrženy identifikátory pro sledování výkonnosti vybraných procesů, ale i transformací ZF V-modelu do ZF W-modelu. Bylo zkoumáno, zda nově simulovaný model povede k optimalizaci procesů ve firmě ZF. Došlo se k závěru, že modely jsou rozdílné zejména v grafickém znázornění. Ve W-modelu jsou schematicky zakresleny procesy, které jsou v obecném V-modelu zahrnuty v detailnějším popisu a ZF V-modelu jsou implementovány ve vývojových diagramech. Jedná se zejména o kontrolní činnost a přípravu na testování. V obecném V-modelu výslovně kontrola zakreslena není. Další odlišností mezi modely je okamžik vytvoření dokumentu „specifikace testování“. V ZF W-modelu je dokument sepsán již ve druhém procesu na rozdíl od ZF V-modelu, kde je dokument vytvořen mnohem později. Ačkoliv je vyhotovena „specifikace testování“ časně, je vždy použita jako vstup až v posledním procesu v podsystému softwarového vývoje. To může zapříčinit časovou prodlevu, při které se pracovníci znovu seznamují s obsahem dokumentu. Ve V-modelu se se specifikací pracuje vzápětí po jejím schválení. Hlavní nevýhodou oproti W-modelu zůstává pozdější zpětná vazba vývojářům na souvisejícím pracovním kroku vývoje.

Modely byly porovnány i z ekonomického hlediska. Pozornost byla věnována finančním prostředkům vynaložených na jednotlivé procesy. Celková výše nákladů je v obou typech modelů srovnatelná.

Na základě výsledků analýzy všech hledisek a rizik, kterými jsou hlavně téměř stejná ekonomická bilance a složitost W-modelu oproti V-modelu, a také po navržení implementace pracovního kroku „Příprava na testování“ do procesu „Analýza softwarových požadavků“ v ZF V-modelu se dochází k závěru, že nový W-model by k optimalizaci procesů ve firmě ZF Engineering Plzeň, s.r.o. nevedl.

Procesy mohou být optimalizovány i využitím vhodného nástroje. Diplomová práce se okrajově zabývala porovnáním zpracování „specifikace testování“ v nástrojích Microsoft Excel a IBM Rational DOORS. Vzhledem ke kladeným nárokům především na dohledatelnost a provázanost požadavků se specifikací testování a s požadavky na změny, je vhodnější použití nástroje DOORS.

# Seznam literatury a informačních zdrojů

[1] PATTON, Ron. *Testování softwaru*. Vyd. 1. Praha: Computer Press, 2002. ISBN 80-722-6636-5.

[2] KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. 1. vyd. Brno: Computer Press, 2004. ISBN 80-251-0342-0.

[3] RICHTA, Karel a Jiří SOCHOR. *Softwarové inženýrství I.* Praha: ČVUT, 1996. ISBN 80-01-01428-2.

[4] SOMMERVILLE, Ian. *Software engineering*. Harlow: Addison-Wesley, 2001. 6. ISBN 0-201-39815-X.

[5] NAIK, Kshirasagar a Priyadarshi TRIPATHY. *Software testing and quality assurance: theory and practice*. Hoboken, N.J.: John Wiley, 2008. ISBN 978-0-471-78911-6.

[6] GERRARD, Paul a Neil THOMPSON. *Risk-based E-business testing*. Boston: Artech House, 2002. ISBN 1-58053-314-0.

[7] WIEGERS, Karl Eugene. *Požadavky na software*. 1. vyd. Brno: Computer Press, 2008. ISBN 978-80-251-1877-1.

[8] BASS, Len, Paul CLEMENTS a Rick KAZMAN. *Software architecture in practise*. 3. vyd. USA: Pearson Education, 2012. ISBN 987-0-321-81573-6.

[9] SVOZILOVÁ, Alena. *Projektový management*. 2. vyd. Praha: Grada Publishing, 2011. ISBN 978-80-247-3611-2.

[10] DOLANSKÝ, Václav, Vladimír MĚKOTA a Vladimír NĚMEC. *Projektový management*. 1.vyd. Praha: Grada Publishing, 1996. ISBN 80-7169-287-5.

[11] NĚMEC, Vladimír. *Projektový management*. 1. vyd. Praha: Grada Publishing, 2002. ISBN 80-247-0392-0.

[12] HÜBNER, Miroslav. *Projektové řízení: příručka manažera*. Praha: Tate International, 2005. ISBN 80-86813-06-1.

[13] HOERMANN, Klaus. *Automotive SPICE in practice: surviving interpretation and assessment*. 1. vyd. Sebastopol: O'Reilly Media, 2008. ISBN 978-1-933952-29-1.

[14] BLECHARZ, Pavel. *Základy moderního řízení kvality.* 1. vyd. Praha: Ekopress, 2011,

122 s. ISBN 978-80-86929-75-0.

[15] IEEE Std 610.12-1990. *IEEE Standard Glossary of Software Engineering Terminology*. New York: IEEE, 1990.

[16] IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications*. New York: IEEE, 1998.

[17] IEEE Std 1471-2000. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. New York: IEEE, 2000.

[18] ANSI/ PMI 99-001-2004. *A Guide to the Project Management Body of Knowledge*. 3. vyd. Pennsylvania: PMI, 2004.

[19] ZELINKA, Bořek. *Testování softwaru* [online]. 2013 [cit. 2013-10-25]. Dostupné z: http://d3s.mff.cuni.cz/teaching/commercial_workshops/previous/1213/zelinka-zajisteni_kvality_softwarovych_produktu.pdf

[20] V-Modell. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-09-22]. Dostupné z: http://de.wikipedia.org/wiki/V-Modell

[21] SPILLNER, A. The W-model – Strengthening the Bond Between Development and Test. [online]. 2002 [cit. 2013-09-22]. Dostupné z: http://cm.techwell.com/sites/default/files/articles/XDD3572filelistfilename1_0.pdf

[22] MUNASSAR, N. a A. GOVARDHAN. A Comparison Between Five Models Of Software Engineering [online]. 2010 [cit. 2013-09-22]. Dostupné z: http://www.ijcsi.org/papers/7-5-94-101.pdf

[23] What is V-model- advantages, disadvantages and when to use it?. *ISTQB Exam Certification* [online]. [cit. 2013-10-03]. Dostupné z: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/

[24] Software design. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-10-27]. Dostupné z: http://en.wikipedia.org/wiki/Software_design

[25] Úvod do funkční bezpečnosti I: norma ČSN EN 61508. *Odborné časopisy* [online].

2014 [cit. 2014-01-21]. Dostupné z: http://www.odbornecasopisy.cz/uvod-do-funkcni-bezpecnosti-i:-norma-csn-en-61508-32520.html

[26] Company Profile. *ZF Friedrichshafen AG* [online]. 2013 [cit. 2013-11-04]. Dostupné z: http://www.zf.com/corporate/en/company/organization/facts_figures/facts_figures.html

[27] *ZF Engineering s.r.o.: Výroční zpráva za rok 2012* [online]. 2013 [cit. 2013-11-04]. Dostupné z: https://or.justice.cz/ias/ui/vypis-sl?subjektId=isor%3a495554&dokumentId=C+14207%2fSL41%40KSPL&klic=en07vz

[28] LUKASÍK, Petr. *Procesní řízení* [online]. [cit. 2013-11-04]. Dostupné z: http://www1.osu.cz/~prochazka/rpri/skripta_ProcesniRizeni.pdf

[29] Interní směrnice a dokumenty ZF Engineering Plzeň, s.r.o.

[30] 8D Report. In: *Ikvalita* [online]. 2007 [cit. 2013-11-10]. Dostupné z: http://www.ikvalita.cz/tools.php?ID=103

[31] LACKO, Branislav. Kvalita softwaru. In: *Odborné časopisy* [online]. 2014 [cit. 2014-02-22]. Dostupné z: http://www.odbornecasopisy.cz/index.php?id_document=31468

[32] KÖHLER, Christian. *Enhancing embedded systems simulation: a Chip-Hardware-in-the-loop simulation framework*. 1. vyd. Wiesbaden: Vieweg Teubner, 2011. ISBN 978-383-4814-753.

[33] Unit testing. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-12-08]. Dostupné z: http://cs.wikipedia.org/wiki/Unit_testing

[34] MISRA C. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-12-10]. Dostupné z: http://cs.wikipedia.org/wiki/MISRA_C

[35] ISO 26262. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-01-08]. Dostupné z: http://en.wikipedia.org/wiki/ISO_26262

[36] ŠMÍDA, Filip. *Zavádění a rozvoj procesního řízení ve firmě*. 1. vyd. Praha: Grada, 2007. ISBN 978-80-247-1679-4.

ZÁPADOČESKÁ UNIVERZITA V PLZNI

**Příloha č. 2**

***děkan / ředitel vysokoškolského ústavu***

V Plzni dne 12.5.2014
č.j. .....

## Rozhodnutí o zveřejnění kvalifikační práce

Na základě návrhu vedoucího kvalifikační práce ze dne 12.5.2014 jsem rozhodl o tom, že kvalifikační práce studenta EVA ANDRLOVÁ s názvem ANALÝZA PROCESNÍHO ŘÍZENÍ se **zveřejňuje / zveřejní po uplynutí stanovené doby / nezveřejní**.

### Odůvodnění

Dne 12.5.2014 jsem obdržel návrh vedoucího výše specifikované kvalifikační práce zpracovaný podle čl. 3 odst. 2 směrnice rektora č. 26R/2013 Zveřejňování kvalifikačních prací.

V tomto návrhu je uvedeno, že kvalifikační práce nemůže být zveřejněna, a to s ohledem na:

1. zákon o ochraně utajovaných informací,
2. předpis upravující právo duševního vlastnictví,
3. smluvní závazek Západočeské univerzity v Plzni ..... *smlouva č. ..... ze dne* .....
4. jiný důvod ..... *specifikovat důvod* .....

Na základě posouzení důvodů uvedených v návrhu jsem v souladu s čl. 3 odst. 2 směrnice rektora č. 26R/2013 Zveřejňování kvalifikačních prací rozhodl o:

zveřejnění / zveřejnění po uplynutí 5 let / nezveřejnění výše specifikované kvalifikační práce.

.....
děkan / ředitel vysokoškolského ústavu