Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologii

Bakalářská práce

Kvalita SW – metody pro vytváření testovacích scénářů

Tomáš Bobek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Bobek

Informatika

Název práce

Kvalita SW – metody pro vytváření testovacích scénářů

Název anglicky

Quality software – methods for creating test scenarios

Cíle práce

Hlavním cílem BP je porovnat současné postupy při vytváření testovacích scénářů pro ověření kvality SW.
Dílčí cíle jsou:
- analyzovat přístupy k hodnocení kvality SW,
- charakterizovat metody pro testování,
- prakticky ukázat tvorby testovacích scénářů na základě UML diagramu,
- zhodnotit pomocí metod vícekriteriálního rozhodování vytvořené scénáře,
- formulovat obecné i specifické závěry.

Metodika

K teoretické části bude využita odborná literatura, publikované odborné a vědecké články.

V praktické části budou na základě UML diagramu vytvořeny testovací scénáře a pomocí metod vícekriteriálního rozhodování bude rozhodnuto, který způsob tvorby scénářů se jeví jako nejvýhodnější.

Pro názorné ukázky tvorby testovacích scénářů bude využit program HP Application Lifecycle Management.

Na základě teoretické a praktické části budou formulována závěrečná doporučení.

Doporučený rozsah práce

35

Klíčová slova

Kvalita SW, Testování SW, Testovací scénář, Software, Specifikace, UML, HP Application Lifecycle Management

---

Doporučené zdroje informací

PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.
ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.
STEPHENS, Matt a Doug ROSENBERG. Testování softwaru řízené návrhem. Brno: Computer Press, 2011. ISBN 978-80-251-3607-2.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

doc. Ing. Zdeněk Havlíček, CSc.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 18. 10. 2016

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 10. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 06. 03. 2017

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Kvalita SW – metody pro vytváření testovacích scénářů" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2017 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Zdeňkovi Havlíčkovi, CSc. za poskytnuté cenné rady a velmi užitečné připomínky. Dále za ochotu a čas strávený na konzultacích.

Kvalita SW – metody pro vytváření testovacích scénářů

Souhrn

Bakalářská práce se zabývá problematikou testování v průběhu vývoje softwaru. V teoretické části jsou popsány metodiky a hlavní pojmy, které jsou spjaty s testováním softwaru. V další části je přiblížena problematika volby přístupu ke psaní testovacích scénářů. Praktická část se zabývá tvorbou testovacích scénářů za účelem otestování aktivity v aplikaci zobrazené pomocí UML diagramu. Na základě poznatků je příklad zhodnocen z několika pohledá zvláště z pohledu časové náročnosti.

Klíčová slova: kvalita SW, testování, testovací scénář, software, specifikace, UML, testovací případ

Quality software – methods for creating test scenarios

Summary

This thesis deals with the issue of testing during software development. The theoretical part describes the methodology and the main concepts that are associated with software testing. The next part focuses on the problematics of access options during the writing test scenarios. The practical part deals with the creation of test scenarios to test the app activity displayed by using UML diagram. An example is evaluated based on the findings from several perspectives especially from the perspective of time consumption.

Keywords: quality SW, testing, test scenario, Software, UML, specification, test case

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V moderní době, kdy hraje kvalita produktu důležitou roli, chce každý z nás mít jistotu, že za peníze, které zaplatil, dostane kvalitní a spolehlivý produkt, při jehož užívání nevzniknou v budoucnu žádné problémy. K těmto produktům patří i bezesporu informační technologie, po kterých roste poptávka každým dnem. Je všeobecně známé, že je mnohem jednodušší předem zabránit problémům, než později řešit jejich důsledky. Kontrola kvality neboli testování je nástroj, který bezesporu umožňuje efektivně jednat tímto způsobem, proto by měli ti, kteří mají s vývojem softwaru co dočinění vynaložit nemalé množství času a prostředků na testování.

O problematice testování softwaru je napsáno nespočet publikací, ale pouze některé z nich lze využít v praxi. I když je pracovník na pozici tester nabyt těmito teoretickými znalostmi, v praxi může být snadno v koncích. Testování softwaru je specifická práce, která je velmi náročná na logiku, trpělivost i pečlivost. Testování doprovází lidstvo již velmi dlouho. Za předchůdce testování je možné považovat zajišťování kvality ve všech technických oborech. Počátky testování začaly vznikat v 50. letech 20. století na Harvardské univerzitě v době, kdy se k provádění složitých výpočtů začaly využívat počítače nulté generace (Mark I a Mark II). V této době se jednalo pouze o debugging, neboli ladění nalezených chyb, který nedokázal odhalit veškeré bugy.

Téma Testování SW jsem si vybral, protože mě tato problematika zajímá a chtěl bych si prohloubit své získané znalosti díky několikaleté praxi na pozici Tester bankovních aplikací na projektech v České spořitelně. I když se nyní věnuji převážně řízení oddělení, které dodává na projekty lidské zdroje a testování jen okrajově, můj zájem o testování dále trvá.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem BP je porovnat současné postupy při vytváření testovacích scénářů pro ověření kvality SW.

Dílčí cíle jsou:

- Analyzovat přístupy k hodnocení kvality SW,
- Charakterizovat metody pro testování,
- Prakticky ukázat tvorby testovacích scénářů na základě UML diagramu,
- Zhodnotit pomocí metod vícekriteriálního rozhodování vytvořené scénáře,
- Formulovat obecné i specifické závěry.

2.2 Metodika

K teoretické části bude využita odborná literatura, publikované odborné a vědecké články.

V praktické části budou na základě UML diagramu vytvořeny testovací scénáře a pomocí metod vícekriteriálního rozhodování bude rozhodnuto, který způsob tvorby scénářů se jeví jako nejvýhodnější.

Pro názorné ukázky tvorby testovacích scénářů bude využit program HP Application Lifecycle Management.

Na základě teoretické a praktické části budou formulována závěrečná doporučení.

3 Teoretická východiska

3.1 Obecný úvod – Co je a co není testování softwaru

Testování softwaru lze pochopit jako řízené spouštění naimplementovaného software s cílem zjistit, zda splňuje implicitní či specifikované potřeby zákazníka (potřeby zákazníka bývají obvykle sepsány ve funkční dokumentaci). Jedná se o zkoumání softwarového produktu a získávání informací o jeho kvalitě a stavu. Hlavním posláním testování je zlepšení kvality SW, bohužel testování samo o sobě ji nezaručuje, pouze poskytuje vstupy pro další rozvoj.

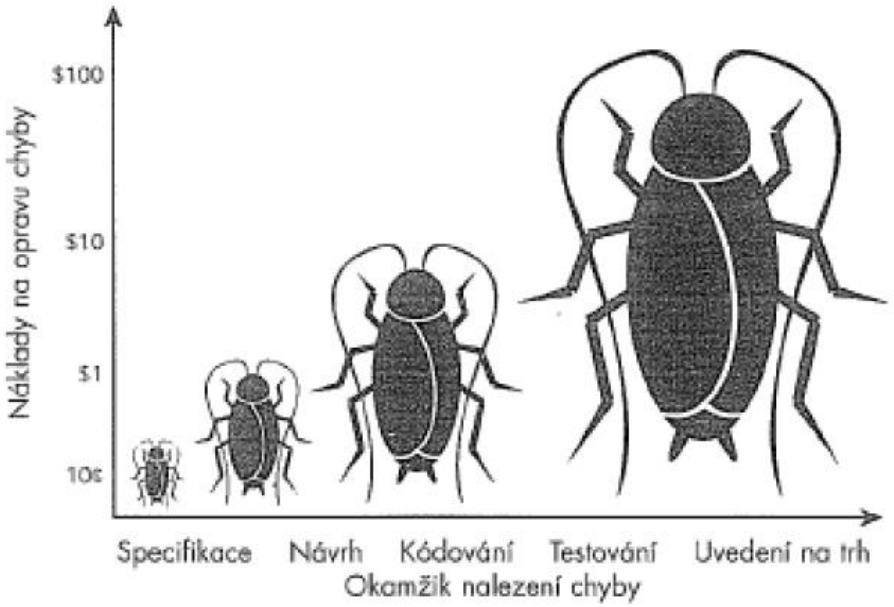
Vnímání okolí je takové, že se jedná o pouhé spouštění softwaru, o pouhé procházení a plnění testovacích scénářů, o hledání defektů a chyb. Mnozí si o testování myslí, že jde pouze o tzv. „klikání“. Ono samotné testování je daleko rozsáhlejší. Existuje několik typů - jak manuálního, tak automatického testování. K testování patří i aktivity jako je řízení, plánování, výběr testovacích případů, navržení testovacích podmínek, kontrola a vyhodnocování výsledků a samozřejmě reportování.

3.2 Proč testovat?

Vývoj SW je neustále zrychlován a požadavky na kvalitu minimálně jsou stejné, ne-li vyšší než tomu bylo před lety. Testování SW tím pádem zvyšuje svou smysluplnost a nepopiratelnou efektivnost, neboť čím dříve jsou chyby odhaleny, tím nižší jsou náklady na jejich odstranění. (Čermák, 2009, online) Toto tvrzení je vyobrazeno na Obrázku 1, kde je názorně ukázán obecně známý graf, který zachycuje poměr nákladů vzhledem k času nalezení chyby.

V literatuře je možné se dočíst například následující:
„Testování softwaru je velice důležitá práce. Vzhledem k rozsahu a složitosti dnešního softwaru je profesionální testování naprosto nezbytné. Rizika jsou příliš vysoká. Nepotřebujeme již žádné vadné počítačové čipy nebo ztracené sondy, které měli přistát na Marsu“. (Patton, 2002) Autor zde naráží na pochybení firmy Intel při testování v průběhu výroby procesoru Pentium v roce 1994, který měl problém s pohyblivou řádovou čárkou při dělení a problému národního úřadu pro letectví a astronautiku (NASA), kdy 3. prosince 1999 se při pokusu o přistání na povrch planety Mars, zcela ztratila sonda, která letěla červenou planetu prozkoumat.

Obrázek 1 Zobrazení nákladu na závislosti čase nalezení chyby.



Zdroj: Patton 2002[1]

3.3 Měření kvality softwaru

Model kvality softwarového produktu [ISO/IEC 2009a] rozděluje atributy kvality systémových výrobků do osmi charakteristik: funkční přiměřenost, účinnost, schopnost spolupráce, použitelnost, bezporuchovost, bezpečnost, udržovatelnost a přenositelnost.

Celková kvalita systému je složena z celkové jakosti systémových prvků a jejich vzájemného působení. Mezinárodní standard [ISO/IEC 2011b] se zaměřuje na kvalitu softwaru jako součást systému. Kontrolou kvality softwaru je schopnost softwarového produktu uspokojovat předem stanovené potřeby za předpokladu, že je užíván za předem stanovených podmínek. Model kvality softwarového produktu, který je uveden v ISO/IEC 9126-1 [ISO/IEC 25010], určuje osm charakteristik jakosti (viz Tabulka 1).

[1] Chyba je zde reprezentována broukem, neboť význam pojmu bug lze z anglického jazyka přeložit právě jako brouk, bacil, štěnice.

Tabulka 1 Definice charakteristik kvality SW.

Definice charakteristik kvality SW	
Funkční přiměřenost	Do jaké míry produkt zabezpečuje funkce, které splňují stanovené a předpokládané potřeby, je-li produkt užíván za předem stanovených podmínek
Účinnost	Výkonnost ve vztahu k výši prostředků využívaných za předem stanovených podmínek
Schopnost spolupráce	Do jaké míry dva (příp. více) systémy nebo jejich součásti dokáží vyměňovat informace a vykonávat své požadované funkce při sdílení stejného HW nebo SW prostředí
Použitelnost	Do jaké míry výrobek může být používán určenými uživateli k dosažení stanovených cílů účinně, efektivně a uspokojivě ve stanoveném kontextu používání
Bezporuchovost	Do jaké míry systém nebo jeho součást plní stanovené úkoly v rámci stanovených podmínek na určitou dobu
Bezpečnost	Do jaké míry jsou informace a data chráněna tak, aby nedošlo k neoprávněné úpravě nebo přístupu neoprávněnou osobou nebo systémem, a neoprávněným osobám či systémům byl odepřen přístup
Udržovatelnost	Stupeň účinnosti a efektivity, s nimiž může být produkt upravován
Přenositelnost	Do jaké míry může být systém nebo jeho součást účinně a efektivně převeden z jednoho HW, SW nebo jiného prostředí do druhého

Zdroj: Autor práce[2]

Kvalita SW z pohledu užití souvisí dle [ISO/IEC 2006a] s uplatněním SW v jeho operačním prostředí, k provádění konkrétních úkolů podle konkrétních uživatelů.

Model jakosti slouží jako rámec, jehož cílem je zajistit, že všechny náležitosti kvality jsou posuzovány z pohledu vnitřní kvality, vnější kvality a kvality užití. (Příbrský, 2012, online)

[2] Tabulka vychází z textu literatury: VANÍČEK, Jiří. 2010. Information systems quality rating. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta. s. 27-32. ISBN 978-80-213-2062-8.

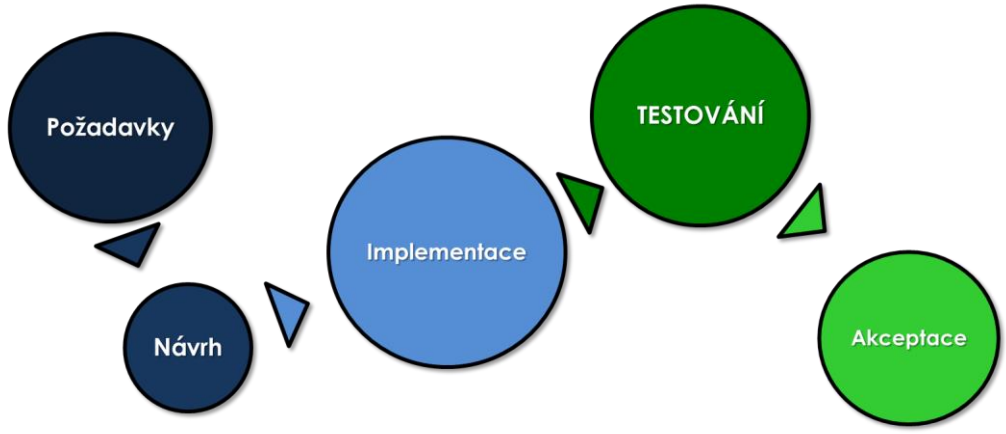
3.4 Kontext testování

3.4.1 Obecný kontext testování

Testování je součástí životního cyklu aplikace. Obecný průběh takového cyklu je:
Požadavky → Analýza → Návrh → Implementace → Testování → Akceptace
V první fázi vývoje SW je potřeba si ujasnit veškeré funkční i nefunkční požadavky, které má náš systém (aplikace) splňovat. Mezi tyto požadavky obvykle patří design, funkce, rychlost, spolehlivost, integrace, … atd. Ve zkratce v této fázi sesbíráme požadavky na aplikaci od zadavatele (zformulujeme vzniklou poptávku na SW). Následně postupujeme k další fázi, ve které analyzujeme zformulované požadavky. Výstupem z této fáze vznikne konceptuální model[3], podle kterého můžeme vytvořit první návrh neboli implementační model. Z hotové analýzy můžeme vytvořit první návrh implementace, který je nezbytný pro vstup do implementační fáze. Po úspěšném návrhu implementace je možné začít se samotnou implementací. Již v této fázi je možné provádět některé testy (např. jednotkové testy). V této fázi je obvykle vytvářena dokumentace systémů. Testování je předposlední fází vývoje systému. Po implementační fázi se může začít s testováním výsledného systému, čímž dojde ke zjištění, zda systém splňuje veškeré požadavky, které byly na samotném začátku stanoveny. Pokud se ukáže, že systém obsahuje nějaké chyby, případně některá funkčnost není naimplementována, je nutné tyto chyby reportovat a následně odstranit (v této fázi spíše doimplementovat). Poslední fází vývoje SW je finální akceptace hotového systému zákazníkem. Provádí se UAT testy[4], a pokud je zákazník spokojený s výsledným produktem, převezme ho a vývoj je u konce. (Foršt, 2016)[5]

[3] Model, který zobrazuje funkční a informační potřeby zadavatele, na jejichž základě může odrážet budoucí potřeby.
[4] Uživatelské Akceptační Testy – testování na straně zákazníka.
[5] Za cenné informace k mé bakalářské práci, které jsem získal na přednášce Metodika testování, vděčím panu Bc. Jiřímu Forštovi, MSc.

Obrázek 2 Obecný kontext testování.



Zdroj: Autor práce

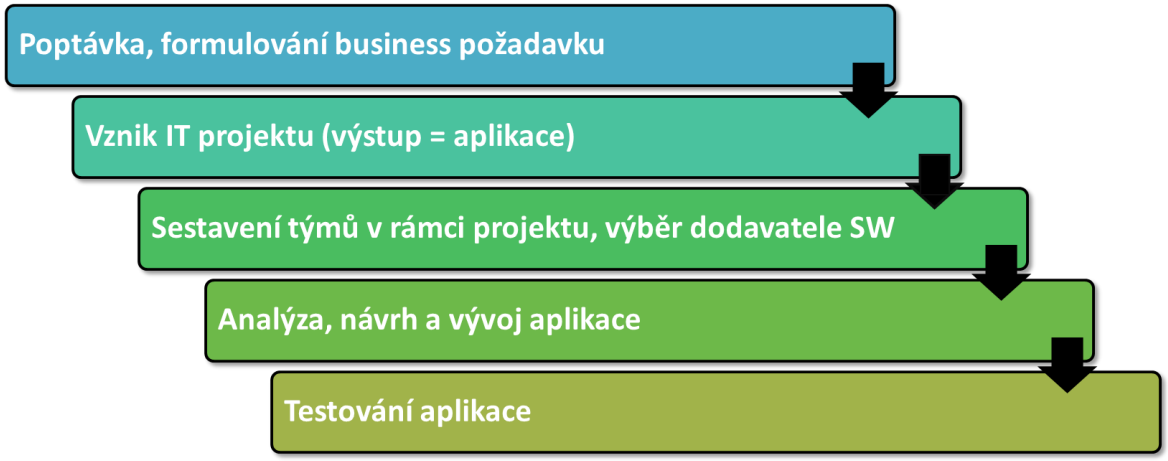
3.4.2 **Business kontext**

V této části se budu na danou věc dívat z pohledu banky. Na počátku je většinou z hlediska businessu snaha o rozšíření nabídky služeb či zlepšení stávajících nabídek. Bance to přináší:

- Nalákání nových zákazníků
- Zvýšení spokojenosti těch stávajících
- Zlepšení interních stavů a procesů
- Vyšší zisky pro banku
- Vyšší efektivita interních procesů

Z hlediska fáze realizace nového produktu nejprve dojde k formulaci business požadavku, a pokud je tento požadavek založen na využití IT, dochází ke vzniku IT projektu, jehož hlavním cílem je ve většině případů vytvořit určitou komplexní aplikaci. V rámci uskutečnění nového IT projektu se sloučí všechny požadavky, které by měla nová aplikace mít. Následuje výběr dodavatele (volba mezi interním týmem, nebo externí firmou), který aplikaci dle specifikovaných požadavků vyvine a dodá do banky. Následně proběhne testování toho, zda jsou všechny požadované funkce korektně zpracovány a zapracovány do aplikace, opět se zde volí mezi interním týmem testerů, nebo externí firmou, která dodá na projekt své lidské zdroje. Pokud proběhne testování bez problémů, nová verze aplikace se může vypustit do ostrého provozu (nasazením aplikace do ostrého provozu však testování aplikace zdaleka nekončí). (Foršt, 2016)

Obrázek 3 Business kontext testování.



Zdroj: Autor práce

3.5 Cíle testování

Hlavní cíl testování by se dal specifikovat mnoha definicemi, ovšem po shrnutí všech definicí se naskytuje pouze jedna:

„*Cílem testování je ověření, zda se aplikace chová podle sepsané specifikace.*“

Pokud tomu tak není, je třeba upozornit vývojáře na odlišné chování aplikace vypsáním tzv. bugu. Po upravení vývojářem chování aplikace je následně povinností testera tuto opravu zkontrolovat. Dalším cílem testování je zjištění stability systému a jeho částí. Primárním cílem testování není hledat za každou cenu chyby/bugy – hledání chyb/bugů je pouze součástí procesu testování, primárně jde však o to, aby aplikace fungovala dle definované specifikace. (Foršt, 2016)

3.6 Základní typy testů

V této kapitole se budu zabývat základními typy a úrovní testování a testů. Mezi tyto úrovně testování a testů patří:

- Akceptační testování (UAT)
- Systémové testování
- Regresní testování
- Konfirmační testy
- Integrační testování
- Smoke a sanity testy

- Jednotkové testování (UNIT testy)
- Factory testy (FAT)

3.6.1 **Akceptační testování**

Akceptační testování neboli tzv. UAT (User acceptance test) jsou akceptační testy na straně zákazníka. Ve zkratce to znamená, že pokud všechny předchozí etapy testování dopadly bez větších nedostatků, je možné předat aplikaci zákazníkovi. Zákazník si právě většinou ještě se svým týmem provede tyto akceptační testy. Jsou prováděny podle předem připravených scénářů, které si společně připravil zákazník s dodavatelem. Testy jsou prováděny na testovacím prostředí zákazníka. Pokud se v aplikaci vyskytují nějaké nedostatky, jsou reportovány zpět vývojovému týmu. Opravené chyby jsou nasazeny zpět na testovací prostředí zákazníka. V této fázi testování je zřejmě nejdůležitější, definovat si předem jakou formou bude probíhat oznamování defektů od zákazníka a jak zajistit opravení těchto defektů v co možná nejkratším čase. Ze zkušenosti je známo, že zákazník zpravidla očekává určitou chybovost aplikace a je spokojen, když jeho testovací tým na nějakou chybu narazí. Velmi nespokojen bývá ve chvíli, kdy je takovýchto defektů v aplikaci nespočet, nebo když se jedná o defekty, které přímo zasahují do funkčnosti celé aplikace. Tyto nalezené chyby je potřeba co nejrychleji opravit a nasadit zpět na testovací prostředí zákazníka, neboť velké prodlevy mezi nalezením a opravením těch chyb má za následek zpoždění termínů nasazení aplikace do ostrého provozu. Tato situace může mít kolosální dopad na úspěch celého projektu. (Hlava, 2011, online) Netestuje se pouze SW produkt, ale i případné další součásti (manuál, šablony dokumentů atd.). Uživatel nehledá přímé defekty, ale prochází scénáře, které zahrnují nasimulované běžné aktivity z reálného provozu.

3.6.1.1 Typy UAT testování

Tabulka 2 Typy UAT testování.

Typy UAT testování	
Alfa testování	Jedná se o zlepšení kvality produktu a zajištění připravenosti na beta testování. Provádí se na konci procesu vývoje, kdy je SW v téměř finální fázi použitelnosti. Trvá 3-5x déle než beta testování a prochází velkým počtem testovacích kol. Odstraňuje většinu kritických chyb a selhání.
Beta testování	Zahrnuje vstupy od zákazníka a připravuje SW na vypuštění do provozu. Provádí se těsně před nasazením SW do ostrého povozu. Trvá většinou jen pár týdnů (výjimečně pár měsíců) a prochází jen několika testovacími koly. Má za úkol odstranit poslední chyby a selhání, které neodhalilo alfa testování.

Zdroj: Autor práce[6]

3.6.1.2 Akceptační hranice

Dále je potřeba, aby byla splněna předem nastavené akceptační hranice. Akceptační hranice znamená, že po nasazení aplikace se aplikace převezme i s některými chybami, ale musí být předem určeno s jakými a s jakým počtem. Zpravidla se chyby dělí do 4 skupin:

Tabulka 3 Typy chyb.

Typy chyb	
A – Fatal	Jedná se o fatální chybu, která způsobuje selhání systému, nedostupnost systému, znemožnění provádění UAT, důležité funkce nepracují tak, jak jsou popsány, nebo že je narušena integrita dat. U této chyby zpravidla nelze implementovat náhradní řešení tzv. workaround.[7]
B – Critical	Tento druh chyby značí částečnou ztrátu funkčnosti systému, přímý dopad na uživatele a klienty, je možné implementovat náhradní řešení tzv. workaround.

[6] Foršt Jiří. Metodika testování. Přednáška 1. 12. 2016.
[7] Pozn. Workaround - dočasné náhradní řešení, dočasné obejití chyby.

C – Major	Částečná ztráta funkčnosti, některé funkce nepracují, tak jak jsou popsány, je možně snadno implementovat náhradní řešení. Jedná se o chybu, která neovlivní funkčnost aplikace, ale bylo by dobré jí opravit.
D – minor	Funkčnosti jsou k dispozici, jedná se spíše o mírně obtěžující anomálie, funkčnost může být zlepšena, nebo přidána. Ve zkratce se jedná o kosmetické vady systému.

Zdroj: Autor práce

Obecně je nastaven standart akceptační hranice: **A 0x, B 3x, C 10x**. Tento standart znamená, že se aplikace nepřevezme, pokud obsahuje chybu typu A. Může obsahovat 3 chyby typu B, 10 chyb C a lze ho akceptovat s chybami typu D. Lze se setkat i s trochu benevolentnějším standardem, přičemž se vždy vše liší dle konkrétní aplikace, situace a závažnosti daných chyb.

3.6.2 **Systémové testování**

Během systémového testování je aplikace ověřována jako funkční celek. Tento druh testů je z pravidla využíván v pozdějších etapách vývoje SW. Jedná se o ověření aplikace z pohledu zákazníka. Podle sepsaných testovacích případů se simulují různé průchody aplikací, které mohou v praxi nastat. Obvykle probíhají v několika kolech. Chyby, na které tester narazil, jsou opraveny a v dalších kolech jsou opravy těchto chyb opětovně testovány. Součástí systémového testování jsou funkční a nefunkční testy. Tato úroveň testů většinou slouží jako výstupní kontrola SW. Systémové testování by mělo být obsaženo ve všech částech testování. Bez systémového testování by testování SW nemělo žádný smysl, byla by ohrožena bezporuchovost výsledného produktu. (Hlava, 2011, online) Toto testování se neprovádí pouze za účelem nalézt chyby, ale také poskytnout informace o celkové funkčnosti produktu. Při vytváření systémových testu se do testovacích případů zahrnují všechny větve příslušného diagramu dané aktivity. V mé praxi na pozici tester se tyto testy hojně využívaly pro testování nových funkčních, nefunkčních požadavků a změnových požadavků na aplikaci.

3.6.2.1 Typy Systémových testů

Tabulka 4 Typy Systémových testů.

Typy Systémových testů	
Funkční testy	Testují se všechny funkce, které by měly být v aplikaci implementovány.
Testy robustnosti	Program poběží i přes veškeré chyby, které může zamlčet dál.
Testy použitelnosti	Jak je aplikace použitelná pro uživatele.
Testy interoperability	Vzájemná provázanost a spolupráce více aplikací.
Testy spolehlivosti	V jaké míře je možné se spolehnout na bezchybný běh aplikace.
Výkonnostní testy	Testy hraniční zátěže, jak se aplikace vyrovná s extrémní zátěží např. náhlý nárůst aktivních uživatelů.

Zdroj: Autor práce

3.6.3 **Regresní testování**

Regresní testy jsou využívány zejména při opětovné kontrole funkčností nebo vlastností aplikace. Hlavním smyslem tohoto testování je ověření, že provedené změny či implementace nových funkčností nemělo žádný vliv na stávající funkce systému. (Hlava, 2011, online) Nevytváří se nové testovací scénáře, jsou použity již vytvořené z důvodu zachycení případných odchylek mezi aktuálními a dřívějšími výsledky. Regresní testy jsou prováděny na jednotkové, integrační a systémové úrovni testování. Hlavní nevýhodou těchto testů, je to, že prakticky není možné otestovat celý systém znovu po každé změně, neboť je to příliš nákladné. Test analytikem jsou vybrány scénáře, jež pokryjí co nejvíce oblastí, ve kterých by mohla mít provedená změna v systému největší dopad. (Foršt, 2016) Ve zkratce si lze regresní testování představit jako ověření, že jakýkoliv zásah do kódu aplikace nepoškodil již fungující část systému. Tento typ testování je vhodné využít i při opravě některého rozsáhlejšího defektu.

3.6.4 **Konfirmační testování**

Pod tímto pojmem se skrývá znovu otestování nalezené chyby testerem, neboli ověření, že chyba, kterou tester v aplikaci objevil a vypsal, byla správně programátorem opravena. V praxi se lze setkat spíše s pojmem Retest, jenž značí právě konfirmační

testování. Jak již bylo výše popsáno, při opravě rozsáhlejšího defektu je vhodné využít regresní testování.

3.6.5 **Integrační testování**

Ve chvíli, kdy je vývojářský tým hotov se svými testy, přichází na řadu testovací tým, neboť integrační testy nepřipravuje programátor, nýbrž testovací tým. V některých případech bývá toto testování označováno jako „testy vnitřní integrace“. Je ověřována bezchybná komunikace mezi jednotlivými komponentami uvnitř aplikace. (Hlava, 2011, online) Dále se testuje integrace 2 a více jednotlivých aplikací vzájemně provázaných. Např. Jedna aplikace využívá určité funkcionality a prvky z aplikace druhé. Cílem tohoto ověřování je objevit chyby vzniklé spojením a následnou integrací více modulů či systémů a jejich rozhraní. (Foršt, 2016) Kupříkladu znovu využiji systémy v bance, kde jsou téměř všechny aplikace provázány s ostatními. Při ověření, zda data z jedné aplikace se propíší do další aplikace, lze využít příklad známé aplikace CRM[8]. Zde je potřeba ověřit zda se data týkající se klienta zobrazí v aplikaci, kterou testujeme. Hlavní nevýhodou integračního testování je fakt, že nelze jednoznačně určit konkrétní příčinu, proč testovaný celek nefunguje.

3.6.5.1 Základní rozdělení

Tabulka 5 Základní rozdělení Integračních testů.

Základní rozdělení Integračních testů	
Intersystémové	Integrace 2 a více systémů do jednoho celku.
Intrasystémové	Integrace modulů do funkčního celku.

Zdroj: Autor práce

[8] Customer relationship management – dá se říci, že se jedná o databázi, ve které se shromažďují, zpracovávají a využívají informace o zákaznících/klientech.

3.6.5.2 Způsoby integračního testování

Tabulka 6 Způsoby integračního testování.

Způsoby integračního testování	
Velký třesk	Integrování všech modulů najednou. Při využití tohoto způsobu je obtížné nalézt problém.
Shora-dolů	S testováním se začíná hierarchicky nejvyšším modulem a přidávají se postupně nižší moduly.
Zdola-nahoru	S testováním se začíná hierarchicky nejnižším modulem a postupně se přidávají vyšší moduly.
Kombinovaný	Jedná se o kombinaci způsobu Shora-dolů a Zdola-nahoru, základ je tvořen nejvyšším a nejnižším modulem, k němuž jsou přidávány další moduly ze střední úrovně.

Zdroj: Autor práce [9]

3.6.6 **Smoke testy**

Smoke testy jsou využívány ve chvíli, kdy je aplikace ve fázi dokončování a lze ji spustit. Jedná se o krátký test, jehož cílem je ověření, zda je vyvíjená aplikace připravena na další fázi testování. Smoke testy mají za úkol jednoduché ověření, zda hlavní funkce programu pracují, jak je sepsáno v dokumentaci. Hlavními funkcemi jsou myšleny části aplikace, které nejsou příliš upravovány. (Hlava, 2011, online) Úspěšné provedení smoke testů je hlavní podmínkou pro spuštění systémového testování, neboť pokud některé hlavní funkcionality aplikace nepracují, jak by měly, nemá cenu dále testovat. Např. kdyby bylo myšleno hlavní funkcionalitou aplikace část přihlašování a tato funkce by nepracovala správně, nemělo by cenu se pouštět do systémových testů, neboť se do aplikace nebude možné ani přihlásit. V tomto případě Test manažer vrátí verzi vývojářům a testerský tým v testování dále nepokračuje.

[9] Foršt Jiří. Metodika testování. Přednáška 1. 12. 2016.

3.6.7 **Unit testy**

Unit testy neboli testování jednotek se využívá po ověření kódu programátorem. V případě objektově orientovaného programování se z pravidla jedná o testování jednotlivých tříd a metod. Když se hovoří o testování jednotek, je v tomto případě myšlena samostatně testovatelná část aplikačního programu. Testy jsou zapisovány zpravidla ve formě programového kódu. Proto se o obsluhu starají vývojáři. U menších projektů se toto testování nevyplácí, neboť je potřeba provést kompletní refaktoring kódu. I v případě větších projektů se tento typ testů nesetkává s podporou vedoucího projektu. Z těchto důvodu je vhodné se tímto druhem testů zabývat již v rané fázi vývoje aplikace. (Hlava, 2011, online)

3.6.8 **FAT testy**

V případě FAT testů neboli Factory acceptance tests dochází k testování na straně dodavatele, kde je ověřování, zda fungují všechny zákazníkovi požadavky na aplikaci. V průběhu testování nemusí být systém plně integrovaný. Během provádění FAT testů dochází k odhalení největšího počtu chyb ve srovnání s ostatními typy testů. Pro vstup do fáze integračního a systémového testování je nutnou podmínkou správný výstup FAT testů.

3.7 **Testovací scénář**

Testovací scénář je sada několika testovacích případů, které na sebe navazují a tvoří tím skupinu logicky navazujících kroků, přičemž jeho hlavním posláním je otestovat zvolenou funkční oblast. Testovací scénáře jsou využívány hlavně jako:

- Prostředek pro pokrytí funkční oblasti, kterou je potřeba otestovat
- Pomocník pro testery sloužící k rozdělení práce
- Vodítko pro zvolenou funkční oblast
- Prostředek, pomocí kterého lze udržovat přehlednost z hlediska managementu defektů, neboť k němu lze přiřadit nalezené chyby
- Prostředek, pomocí kterého lze zjistit vývoj chybovosti zvolené funkční oblasti aplikace, neboť uchovává svoji historii

Bohužel se o testovacím scénáři nedá říci, že je na 100% věrohodným nástrojem pro testování, a to především ze dvou hlavních důvodů:

- Testovací scénáře píší testeři (resp. test designéři) na základě specifikace, kteří nejsou neomylní a mohou při vytváření udělat chybu.
- V momentě aktualizace specifikace, ne vždy dojde i k aktualizaci scénáře.

Z toho tvrzení tedy vyplývá, že by tester při práci s testovacím scénářem měl mít na paměti, že ne všechny informace a údaje v něm obsažené musí být 100% pravdivé. Pokud je tedy tester na pochybách a přijde mu na scénáři něco divného, je v první řadě potřeba se podívat do aktuální verze specifikace a tento případ překontrolovat. V ideálním případě i chybu ve scénáři opravit. (Foršt, 2016)

3.8 Možné přístupy k návrhu testů

Testovací scénáře jak již bylo výše zmíněno, jsou vytvářeny pro účely pokrytí určité funkční oblasti aplikace, slouží k zjednodušení práce testera a pro reportování výsledků v jaké stavu se nachází testovaná aplikace. Obsah scénáře je tvořen zejména ze dvou částí:

- Popisná část – zde je uvedena oblast testování, kategorie testu, vstupní podmínky pro spuštění scénáře, případně návod, jak daný scénář nasimulovat.
- Seznam po sobě jdoucích testovacích případů neboli kroků, které určují co má tester v aplikaci provést a jaký je očekávaný stav akce. V praxi se používá termín Step.

Při tvorbě scénářů je možné použít dva typy přístupů:

Tabulka 7 Možné přístupy ke psaní testovacích scénářů.

Možné přístupy ke psaní testovacích scénářů	
Procedurální	Při cyklení některé činnosti se vytvářejí ty samé posloupnosti kroků i v ostatních scénářích.
Modulární	Tento přístup lze nazvat jako strukturovaný přístup k tvorbě testů. Stále se opakující aktivity jsou zaznamenány na jednom místě do tzv. bloků a konkrétní testovací scénáře jsou následně z těchto bloků složeny, často při využití různých vstupních parametrů.

Zdroj: Autor práce

Pro oba přístupy tvorby testů lze využít parametry a konfigurace. Při použití parametrů, je zvyšována flexibilita testování tím, že je umožněno opakovaně provádět stejný test pro různá data. Na rozdíl od parametrů má konfigurace za úkol definovat sady dat případně konkrétní prostředí k testovacímu scénáři. To znamená, že je možné nejprve

vytvořit obecný testovací scénář a poté mu definovat sadu dat pro každou zvolenou konfiguraci. (Foršt, 2016)

Tabulka 8 Využití parametrů a konfigurací.

Využití parametrů a konfigurací	
Parametr	Proměnná, ke které může být přiřazena hodnota definovaná mimo testovací případ. Využití parametrů může zvýšit flexibilitu testování tím, že je umožněno provádět stejný test pro odlišná data.
Konfigurace	Představuje specifické užití testovacího případu. Definuje skupinu dat, ve které by měl být testovací případ využit.

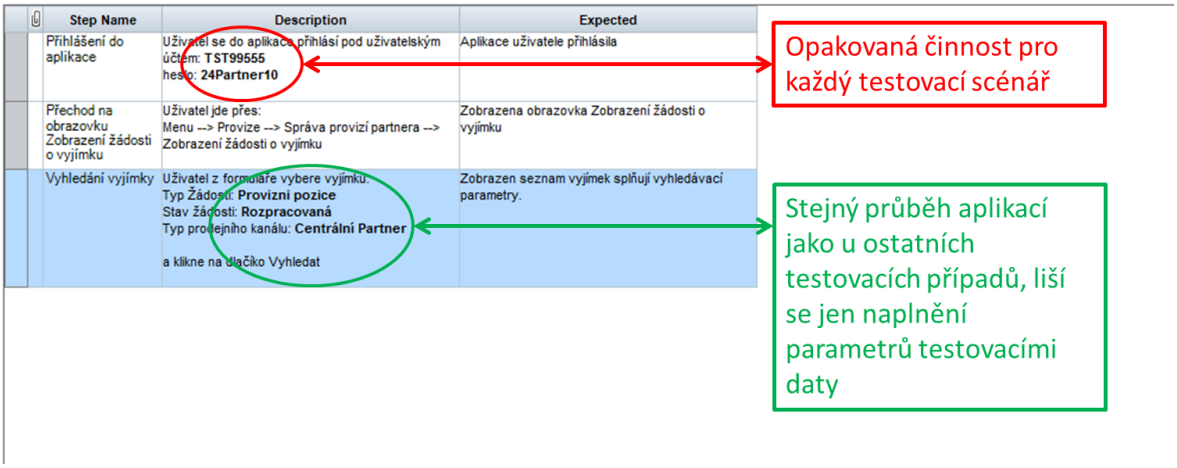
Zdroj: Autor práce

3.8.1 Procedurální vs. modulární způsob psaní testovacích scénářů

V průběhu navrhování testovacích scénářů se používal a často ještě stále používá tzv. procedurální přístup. Jak již bylo popsáno výše, jde o to, že jedna funkčnost, nebo dokonce i celý scénář je popsán do jednoho testovacího případu, ve většině případů i s konkrétními daty. V případě, že se ve scénáři některé činnosti opakují, jsou stejné informace zkopírovány z jednoho kroku do druhého nebo z jednoho testovacího scénáře do druhého.

Příklad:

Obrázek 4 Příklad procedurálního přístupu k návrhu testů.



Zdroj: Autor práce

Další možností pro tvorbu testů je tzv. modulární přístup (mnohdy je též popisován jako strukturovaný přístup k návrhu testovacích případů). Modularita zde reprezentuje skládání testovacích případů z menších částí tak, aby opakovaně provádějící se testovací

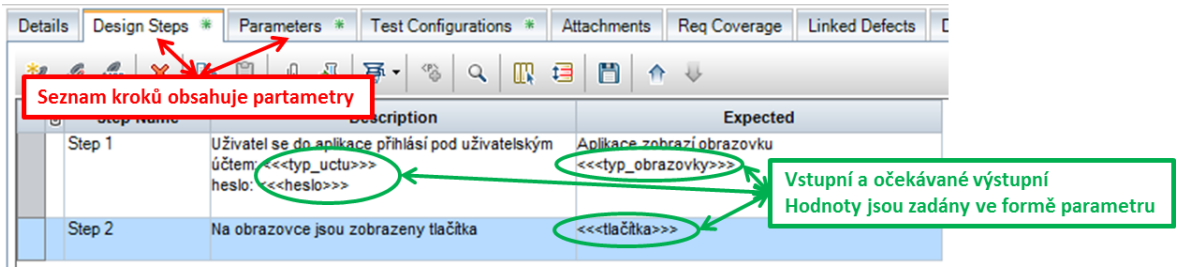
aktivity byly zaznamenány na jednom místě a následně byly složeny a volány z testovacích případů často s využitím různých vstupních parametrů.

Jednotlivé moduly lze následně spojovat dvěma způsoby:

- Vložením jednoho testovacího případu do druhého
- Složením testovacích případů do sad

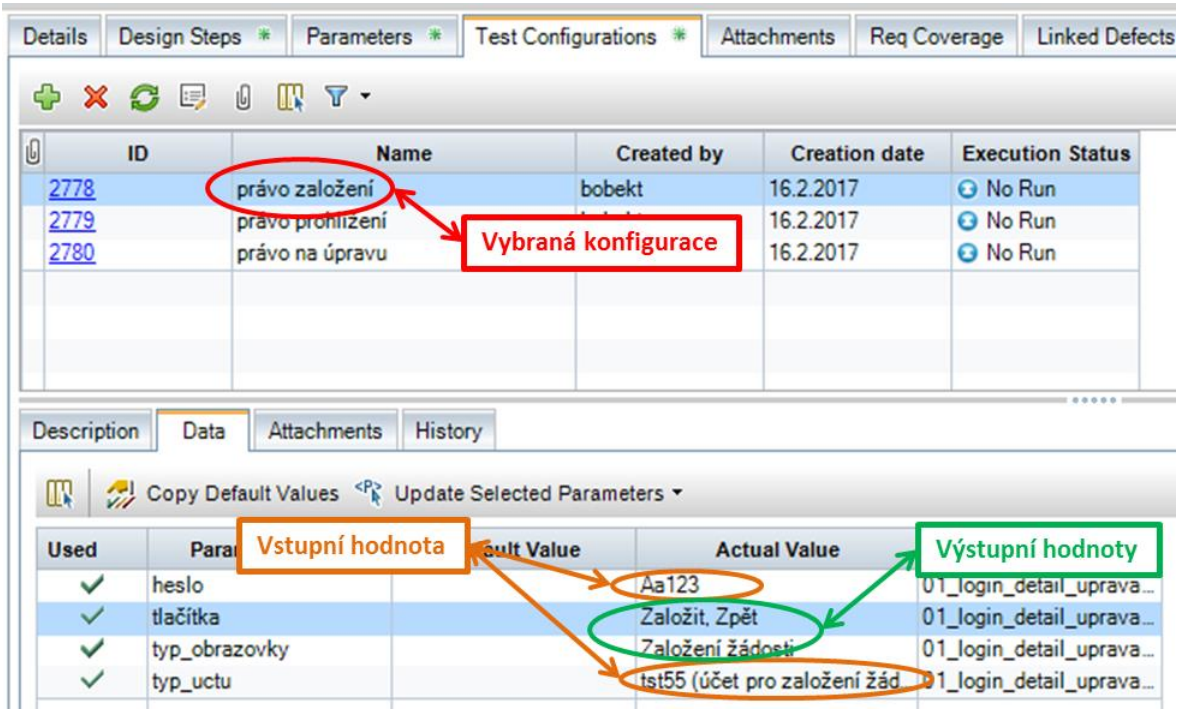
Příklad:

Obrázek 5 Příklad modulárního přístupu k návrhu testů - testovací případ.



Zdroj: Autor Práce

Obrázek 6 Příklad modulárního přístupu k návrhu testů - konfigurace s parametry.



Zdroj: Autor Práce

3.8.2 Výhody modulárního přístupu

Mezi hlavní výhody při využití modulárního přístupu bezesporu patří opakovatelnost ve smyslu, že není nutné popisovat opakující se testovací kroky na více místech. V případě

opakování některé činnosti, se pouze použije již vytvořený testovací scénář s pouhou změnou v parametru či konfiguraci. Pokud se změní některá část aplikace (např. způsob přihlašování), stačí upravit postup testování jen na jednom místě. S tímto se pojí i menší náchylnost testovacích případů k chybám, resp. jejich snadnější oprava. Pokud nastane situace, kdy je špatně popsán testovací případ (např. z důvodu špatně sepsané či pozdě dodané specifikace, nebo z důvodu omylu autora testovacího případu), rovněž tuto situaci stačí upravit pouze na jednom místě. U procedurálních přístupů je testovací scénář popsán od začátku do konce zpravidla různě v závislosti na přístupu autora, zatímco modulární přístup ve většině případů má svá obecná pravidla a tím vyžaduje sjednocení stylu hned v počátku návrhu testů, tudíž výhodou je i větší jednotnost. Výhody lze nalézt i v případě dohledávání problematických oblastí aplikace, kdy jsou reporty odvolávány na konkrétní testovací případy. Jak již bylo zmíněno, další výhodou je i volání testů v kombinaci s parametrizací dat podobných funkčností s různými vstupními údaji a užití na více místech. (Zientková, 2016)

Tabulka 9 Výhody modulárního přístupu.

Hlavní výhody modulárního přístupu	
Opakovatelnost použití	Není nutné popisovat stejné testovací kroky na více místech.
Snadnější údržba	Po změně některé části - stačí upravit na jednom místě.
Menší náchylnost k chybám	V případě chyby ve specifikaci - stačí scénář upravit na jednom místě.
Větší jednotnost	Jsou nastavena obecná pravidla, kterých by se měl držet každý autor testu.
Lepší dohledatelnost chyb	Reporty se odvolávají na konkrétní testovací případy.
Využití v případě podobných funkčností	Obzvláště vhodný při využití v kombinaci s parametrizací dat.

Zdroj: Autor práce

3.8.3 **Využití modulárního a procedurálního přístupu**

Nelze jednoznačně říci, který z přístupů psaní scénářů je správný. Určitě jsou však oblasti, ve kterých se využití modulárního způsobu jeví jako výhodnější, a naopak oblasti, kde je využití procedurálního přístupu zcela postačující. V průběhu testování jedné dílčí části aplikace je možné tyto přístupy kombinovat. Obzvláště v případě, kde funkčnost, na kterou nejsou napojeny další scénáře, lze popsat jedním testovacím scénářem, a zároveň

mít jinou oblast testů popsanou modulárně. Případy využití modulárního a procedurálního přístupu jsou popsány v Tabulce 10.

Tabulka 10 Využití Modulárního a Procedurálního přístupu.

Charakteristika aplikace	Malá šířka aplikace	Velká šířka aplikace
Malá hloubka aplikace	Procedurální přístup, testovací případ pro jednotlivou funkčnost (často technické testy).	Modulární přístup, ideálně s využitím konfigurací.
Velká hloubka aplikace	Procedurální přístup, většinou pro obchodní proces. Aplikace s velmi malým počtem testovacích případů, kdy se neopakují funkčnosti.	Modulární přístup, ideálně s využitím konfigurací.

Zdroj: Zientková, 2016

V Tabulce 10 jsou popsány případy, kdy je vhodné využít procedurální a modulární způsob. Hloubkou aplikace je myšlena délka případů užití (dlouhý/krátký průchod aktivitou) a šířkou aplikace je myšlen počet případů užití neboli počet podobných průchodů aplikací.

V případě užití procedurálního přístupu je vždy vhodné popsat testovanou aplikaci alespoň několika testovacími případy, které jsou následné poskládány do testovacích sad. Tím, že jsou testovací případy vytvořeny, je umožněno sledování jejich užití v testech (kdy byly spuštěny a s jakým výsledkem), napojování incidentů a provádění různých reportů (ovšem v omezené míře, oproti modulárnímu přístupu).

3.8.4 Nároky na autory testovacích případů dle modulárního přístupu

Při vytváření testovacích scénářů modulárním způsobem je kladen větší důraz na kvality autora scénářů, neboť v tomto přístupu je vyžadována hlubší a včasná znalost aplikace, což je skloubeno s včasnou dodávkou a kvalitou podkladů pro testy. Je nutné počítat s delší počáteční dobou na analýzu, na vytváření testovacích případů, ovšem tento fakt se vrátí při následném skládání případů do sad a je především velkou úsporou při úpravách případů pro další verze aplikace. Autor by měl mít jistou míru abstraktního myšlení, neboť testovací případy pro obecné funkčnosti neobsahují konkrétní data. Parametry je možné naplnit při rozlišení konfigurací.

4 Vlastní práce

Po shrnutí teoretických východisek se praktická část bude zabývat ověřením, jaký přístup k vytváření testovacích scénářů se jeví pro daný diagram jako nejvýhodnější.

Pro tyto účely bude využit diagram vytvořený přímo k účelu mé bakalářské práce. Jedná se o UML diagram popisující proces v aplikaci, ve kterém je zobrazena aktivita založení/zobrazení žádosti (viz odstavec 4.2 UML). Vycházíme z předpokladu, že proces je v aplikaci implementován a simulujeme situaci, kdy tester začíná s analýzou diagramu a přistupuje ke psaní testovacího scénáře za účelem otestování každé větvě UML diagramu.

Testovací scénáře budou vytvořeny v aplikaci HP Aplication Lifecycle Management a v neposlední řadě na základě atributů: čas strávený nad analýzou diagramu, doba vytváření scénářů, počet vytvoření scénářů, čas strávený nad úpravou scénářů, doba otestování diagramu, finanční náročnost na vytvoření scénářů, finanční náročnost na otestování diagramu bude pomocí metod vícekriteriální analýzy rozhodnuto, který přístup psaní scénářů se jeví jako nejvýhodnější.

4.1 Aplication Lifecycle Management

V praxi se využívá celá řada programů, které mohou usnadnit testerskou práci. Níže je uvedena hrstka z nich:

- **FireBug** – nejen vývojářský pomocník do prohlížeče Mozilla Firefox
- **IETester** – zobrazování webových aplikací na různých jádrech IE
- **Mantis** – bug reporting
- **JIRA** – bug reporting
- **BugZilla** – bug reporting
- **TestLink** – správa testovacích scénářů
- **IBM Rational Functional** Tester
- **Visual Studio Test Professional**
- **TestArchitect**
- **TestComplete**, **LoadComplete**
- **LoadIU**[10]

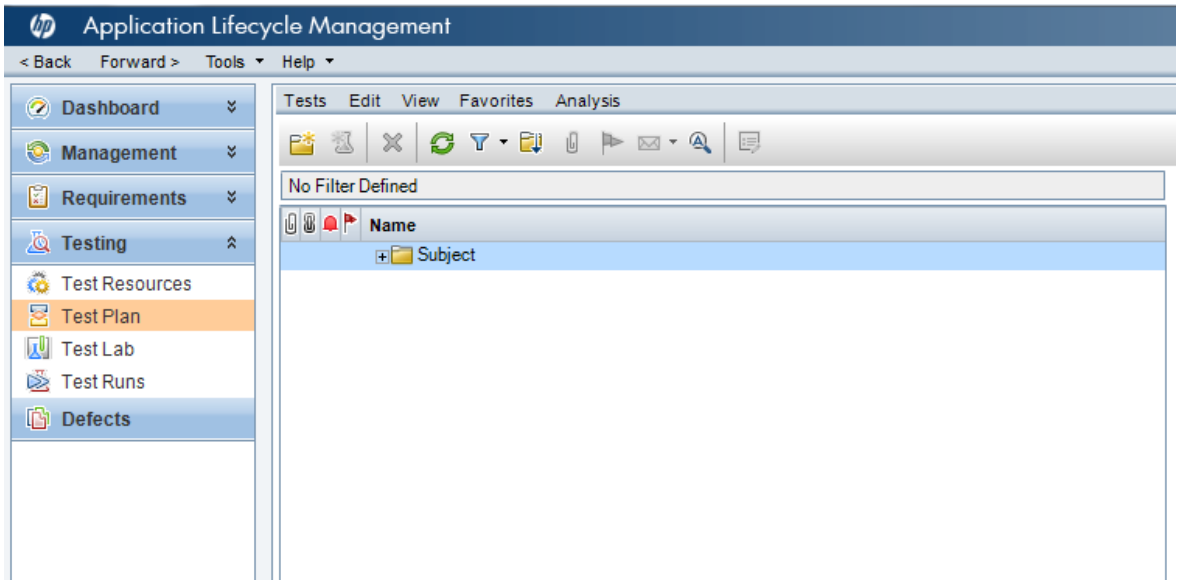
[10] Zdroj: http://testovanisoftwaru.cz/nastroje/

V této bakalářské práci budu, jak je již výše napsáno, využívat program HP Aplication lifecycle Management (dále jen ALM), neboť s ním mám osobní zkušenost a dle mého názoru se řadí mezi nejkomplexnější nástroje pro účely testování softwaru.

Aplikace ALM je sada softwarových produktů firmy HP (Hewlett-Packard) a slouží jako podpůrný nástroj pro proces testování. ALM se dotýká všech fází testovacího procesu od správy požadavků, vytváření a spouštění scénářů, až po reportování chyb, ale také reportování průběhu a výsledků testů.

Pro účely praktické části této bakalářské práce bude využit modul Testing, pro tvorbu manuálního testovacího scénáře, následně podvolba Test Plan (viz Obrazek 7).

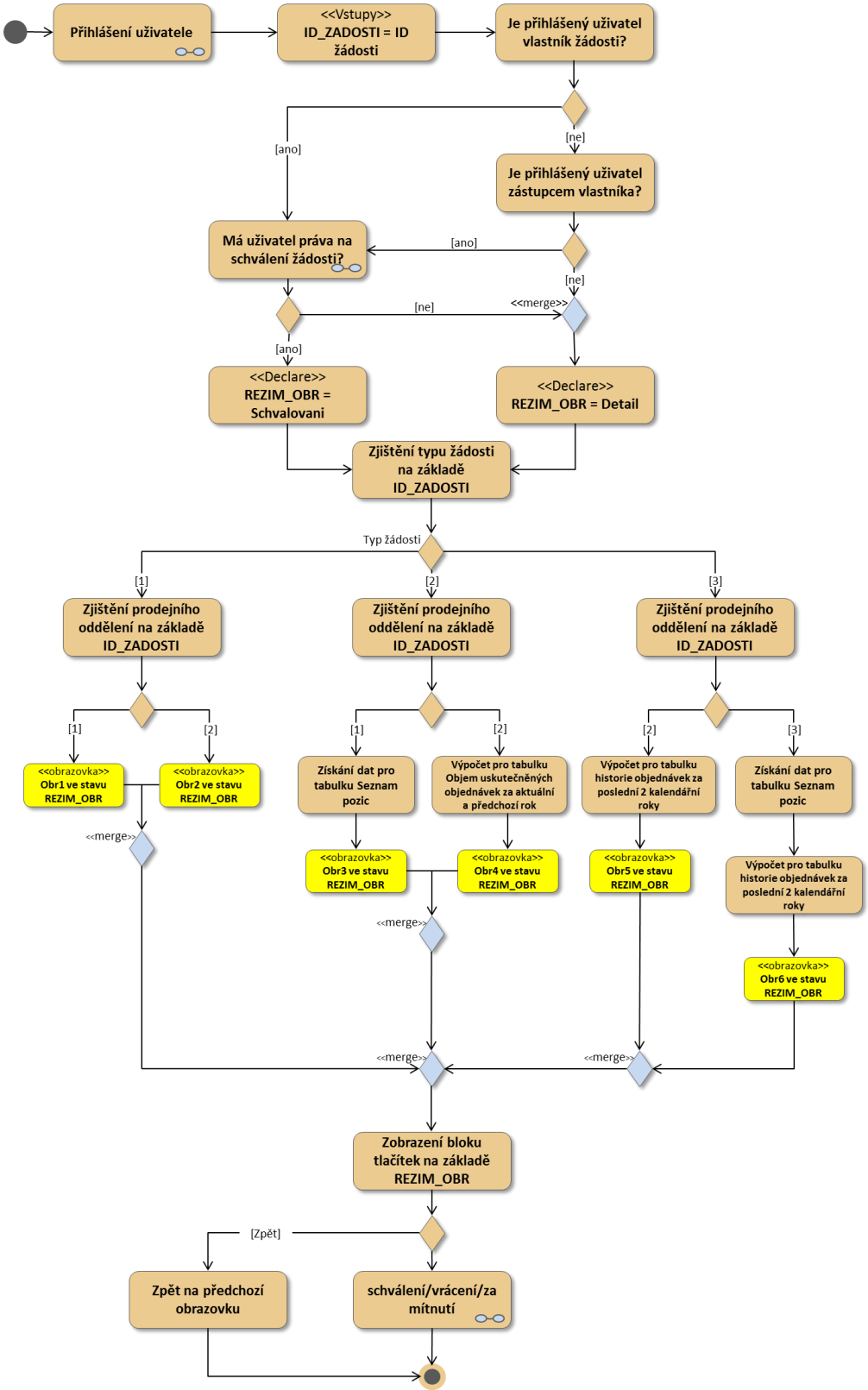
Obrázek 7 Podvolba Test Plan v modulu Testing v aplikaci ALM.



Zdroj: Autor práce

4.2 UML diagram

Obrázek 8 UML diagram vytvořený k účelu praktické části Bakalářské práce.



Zdroj: Autor práce

V praktické části budu pracovat s výše zobrazeným UML diagramem, na který vytvořím testovací scénáře za účelem pokrytí každé větve diagramu. Diagram zobrazuje aktivitu ve smyšlené aplikaci, kde dochází k vytvoření, schválení, zamítnutí a zobrazení žádosti. Diagram simuluje situaci, kdy se uživatel do aplikace přihlásí a chce zobrazit, případně upravit/zamítnout vytvořenou žádost. Na základě práv uživatele a vstupních parametrů jako je typ žádosti a prodejního oddělení se uživateli zobrazí daná obrazovka v příslušném režimu. V případě, že je obrazovka zobrazena v režimu detail, tak uživatel má práva pouze danou žádost zobrazit (zobrazeno pouze tlačítko zpět). Pokud je obrazovka zobrazena v režimu schvalování, uživatel má na danou žádost práva schválení a zamítnutí žádosti (zobrazeny tlačítka zamítnout, schválit a zpět).

4.3 Tvorba scénářů procedurálním způsobem

Jak již bylo výše zmíněno, testovací scénáře budou vytvářeny v aplikaci ALM. Manuální testy se vytváří v modulu Test Plan po stisknutí tlačítka New test. Na obrazovce, která se zobrazí po stisknutí tlačítka New test (Obrázek 9) je potřeba vyplnit pole, která slouží k předání informací testerovi o daném testu. Na základě těchto polí lze zjistit stav testu, část aplikace, kterou test pokrývá, priorita testu a mnoho dalšího. Pro účely praktické části budu vyplňovat pouze pole pojmenování testu a popis testu včetně předpokladů. Předpoklady slouží testerovi jako pomůcka pro spuštění scénáře. Uvádí se zde data, které jsou potřeba si předem připravit, či v aplikaci nastavit. Splnění předpokladů je důležité pro spuštění testu, neboť pokud by předpoklady nebyly splněny, spuštění scénáře by nemělo žádný smysl.

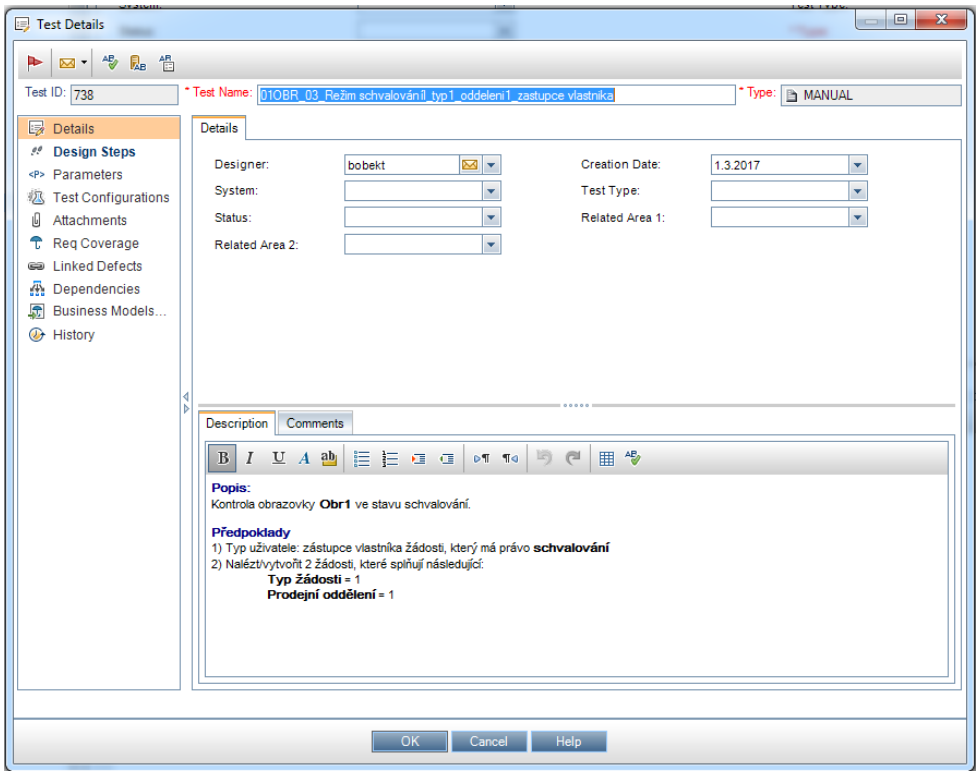
4.3.1 Design testů procedurálním způsobem

Před samotným designem testu by měla předcházet analýza diagramu. V případě psaní scénářů procedurálním způsobem - není pro tuto aktivitu potřeba vynakládat velké úsilí. Ovšem je třeba si zprvu říci, kolik zhruba scénářů bude potřeba napsat pro otestování průchodu každé větve diagramu.

V diagramu jsou 3 možné typy uživatelů (vlastník žádosti s právem na schválení, zástupce vlastníka žádosti bez/s právem na schválení a uživatele, který nemá právo na schválení žádosti). Následuje zobrazení obrazovky v příslušném režimu, dle typu žádosti a typu prodejního oddělení.

Na Obrázku 9 se zobrazeno témata přípravy testovacího scénáře, kde je potřeba uvést krátký popis testu a předpoklady, které je potřeba, aby tester před spuštěním testu splnil. Dále je důležitý i pojmenování testu. Pro ukázku jsem si vybral test, který otestuje případ, kdy přihlášený uživatel je zástupcem vlastníka žádosti, tudíž obrazovka by měla být zobrazena v režimu schvalování, typ žádosti a typ oddělení musí být roven 1. Jako nadpis bylo zvoleno co nejvýstižnější pojmenování testu: „01OBR_03_Režim schvalováníl_typ1_oddeleni1_zastupce vlastnika“

Obrázek 9 Příprava scénáře (procedurální způsob) - 01OBR_03_Režim schvalováníl_typ1_oddeleni1_zastupce vlastnika.



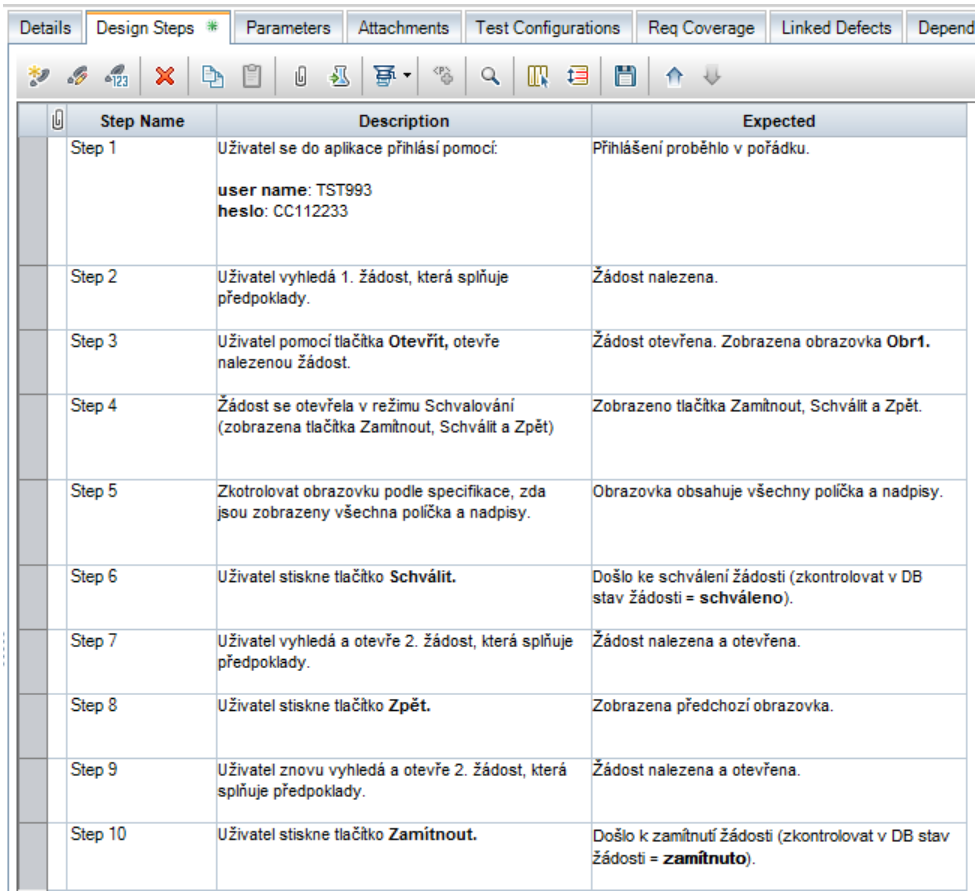
Zdroj: Autor práce

Po vyplnění základní údajů o testu je na řadě již samotný design jednotlivých kroků scénáře. V aplikaci ALM se kroky tvoří v sekci Design Steps. Pomocí tlačítka New Steps vytváříme jednotlivé kroky scénáře. Při přípravě kroků scénáře vycházíme obvykle ze specifikace (v tomto případě diagramu sloužícího pro účely této práce). Vyplňují se zde 3 pole:

- Name - pojmenování kroku (není nutné měnit)
- Description – obvykle akce, kterou by měl tester provést/zkontrolovat
- Expected – obvykle reakce, kterou by měl systém vykonat/zobrazit

Pro příklad vyplnění jednotlivých kroků v této části práce je uveden již zmíněný případ, kdy je testován možný typ přihlášeného uživatelského účtu. Následuje zobrazení obrazovky v příslušném režimu dle typu žádosti a typu prodejního oddělení, ověření funkčnosti schvalovacích tlačítek na obrazovce a kontrola dat v databázi.

Obrázek 10 Kroky testu: 01OBR_03_Režim schvalováníl_typ1_oddeleni1_zastupce vlastnika.



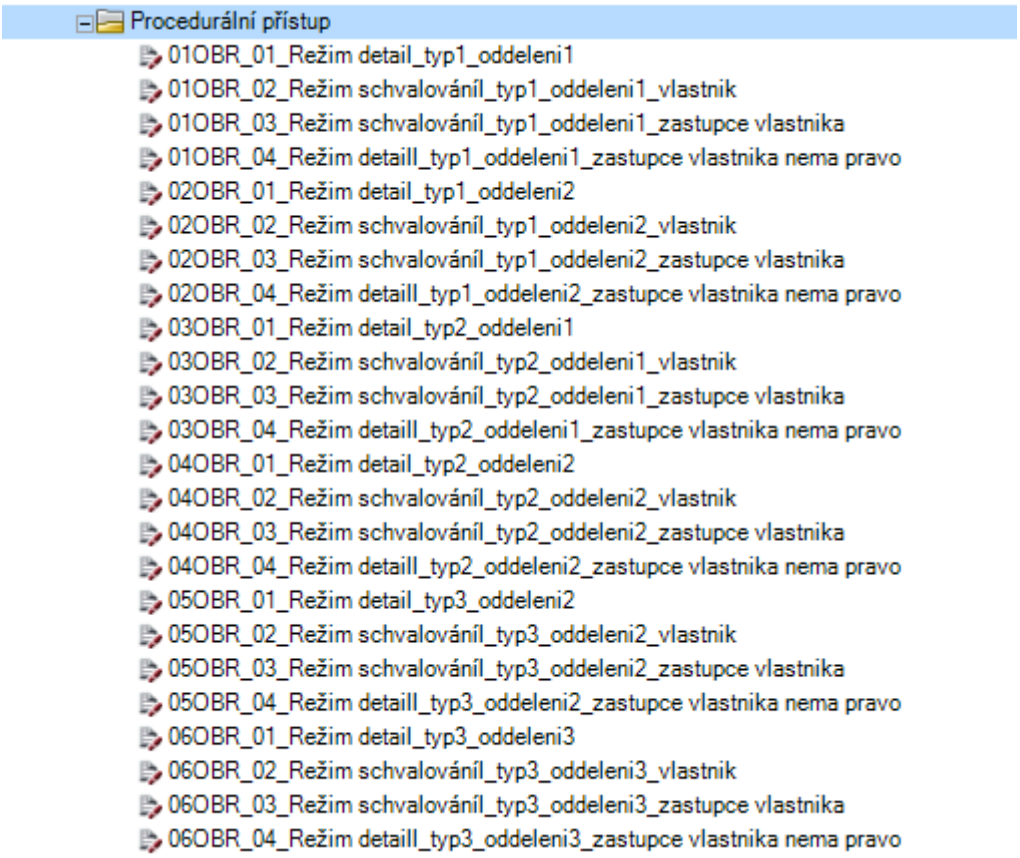
Details | Design Steps | Parameters | Attachments | Test Configurations | Req Coverage | Linked Defects | Depend

	Step Name	Description	Expected
	Step 1	Uživatel se do aplikace přihlásí pomocí: **user name**: TST993 **heslo**: CC112233	Přihlášení proběhlo v pořádku.
	Step 2	Uživatel vyhledá 1. žádost, která splňuje předpoklady.	Žádost nalezena.
	Step 3	Uživatel pomocí tlačítka **Otevřít,** otevře nalezenou žádost.	Žádost otevřena. Zobrazena obrazovka **Obr1.**
	Step 4	Žádost se otevřela v režimu Schvalování (zobrazena tlačítka Zamítnout, Schválit a Zpět)	Zobrazeno tlačítka Zamítnout, Schválit a Zpět.
	Step 5	Zkotrolovat obrazovku podle specifikace, zda jsou zobrazeny všechna políčka a nadpisy.	Obrazovka obsahuje všechny políčka a nadpisy.
	Step 6	Uživatel stiskne tlačítko **Schválit.**	Došlo ke schválení žádosti (zkontrolovat v DB stav žádosti = **schváleno**).
	Step 7	Uživatel vyhledá a otevře 2. žádost, která splňuje předpoklady.	Žádost nalezena a otevřena.
	Step 8	Uživatel stiskne tlačítko **Zpět.**	Zobrazena předchozí obrazovka.
	Step 9	Uživatel znovu vyhledá a otevře 2. žádost, která splňuje předpoklady.	Žádost nalezena a otevřena.
	Step 10	Uživatel stiskne tlačítko **Zamítnout.**	Došlo k zamítnutí žádosti (zkontrolovat v DB stav žádosti = **zamítnuto**).

Zdroj: Autor práce

Pro celkové pokrytí diagramu testy, bylo potřeba napsat 24 testovacích scénářů (viz Obrázek 11) procedurálním způsobem, přičemž se drtivá většina z nich lišila pouze ve změně některých atributů jako přihlašovací údaje uživatele a zobrazená obrazovka.

Obrázek 11 Kompletní seznam vytvořených scénářů procedurálním způsobem.



Zdroj: Autor práce

Na Obrázku 11 je zobrazen kompletní seznam scénářů, které bylo potřeba vytvořit pro celkové pokrytí diagramu procedurálním způsobem. Pro účely této práce nebudou všechny scénáře zobrazeny, neboť se liší pouze ve změně některých atributů od scénáře, který je zobrazen na Obrázku 10.

4.3.2 Příprava testů na spuštění testerem

V této kapitole bude popsána aktivita, která je potřeba v aplikaci ALM vykonat, aby bylo možné vytvořený testovací scénář spustit. Exekuce testů se provádí v modulu Test Lab. Zde je potřeba vytvořit testovací sadu (pomocí tlačítka New Test Set), do které budou přidány vytvořené scénáře.

Scénáře se do testovací sady přidávají pomocí tlačítka Select Tests. Po stisknutí tohoto tlačítka dojde v pravé části aplikace k zobrazení nového okna, kde je zobrazen modul Test Lab. V této části je potřeba vybrat vytvořený scénář a pomocí tlačítka Add Tests to Test Set přidat vytvořený scénář do testovací sady. Tímto způsobem se přidají všechny vytvořené scénáře do testovací sady, odkud je bude moci tester spustit.

Tento způsob přidávání testů do testovacích sad přináší jisté výhody a nevýhody. Hlavní výhodou pro testera je fakt, že všechny kroky scénáře jsou pohromadě a nemusí ke každému modulu diagramu spouštět další scénář. Tudíž otestování jedné větve diagramu může ušetřit značné množství času.

Pro vedení projektu, které je zodpovědné za reportování výsledků testování, může být tento přístup nevýhodou z důvodu malého přehledu, která část diagramu/aplikace je chybová. Neboť pokud tester označí scénář stavem Failed, není na první pohled jasné, kde přesně došlo k chybě nebo který modul aplikace je chybový.

4.3.3 Úprava vytvořeného scénáře procedurálním způsobem

Z důvodu opakujících se kroků v každém scénáři je při změně jakékoliv proměnné potřeba provést úpravu v každém vytvořeném scénáři. V případě, že bylo vytvořeno 24 testovacích scénářů, tester si musí otevřít každý scénář zvlášť a manuálně jej upravit.

4.4 Tvorba scénářů modulárním způsobem

V této části vytvořím znovu testovací scénáře pro pokrytí diagramu testy, jen s tím rozdílem, že nyní budou vytvořeny modulárním způsobem a následně složeny do testovacích sad.

4.4.1 Analýza UML diagramu

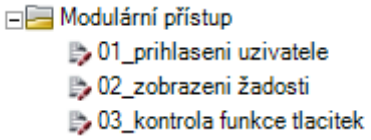
Modulární způsob je typický tím, že před designem testů je potřeba diagram důkladněji zanalyzovat. Jak již název přístupu (modulární) napovídá, je potřeba si diagram rozdělit na tzv. moduly, které se následně spojí dohromady v testovacích sadách, za účelem otestování každé větve diagramu. Jako první modul byla zvolena část, kdy aplikace ověřuje typ uživatele, zda je vlastníkem žádosti, zástupcem vlastníka žádosti s/bez práva schválení, nebo zda je uživatel bez práva na schválení žádosti. Jako další část bylo zvoleno zobrazení typu obrazovky na základě atributů jako je typ žádosti[11] a typ prodejního oddělení[11]. Další, a zároveň poslední částí, byl zvolen modul, kde se uživatel pokusí stisknout tlačítko s příslušnou operací (schválit, zamítnout a zpět).

[11] Možné typy žádosti a typy prodejního oddělení jsou 1,2,3

4.4.2 Design testů modulárním způsobem

Testy modulárním způsobem se tvoří ve stejné části aplikace ALM jako procedurální, tudíž v sekci Test Plan, pomocí tlačítek New Test a New Step. Před samotným designem bylo stanoveno, že diagram bude rozdělen do 3 částí neboli modulů, proto bude potřebovat vytvořit 3 testovací scénáře (viz Obrázek 12).

Obrázek 12 Seznam scénářů modulárním způsobem.



Zdroj: Autor práce

Každý scénář bude mít za úkol otestovat jeden modul diagramu. (viz Tabulka 11)

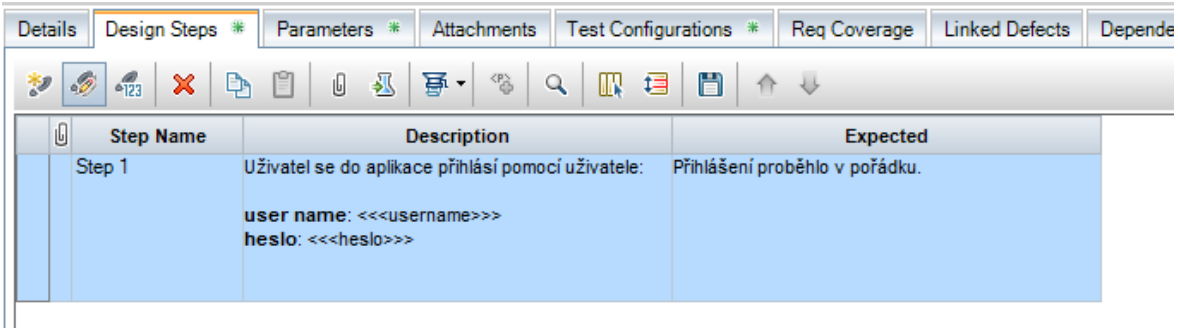
Tabulka 11 Popis vytvořených scénářů modulárním způsobem.

Popis vytvořených scénářů modulárním způsobem	
01_prihlaseni uzivatele	Testování přihlášení uživatele
02_zobrazeni žadosti	Testování zobrazení žádosti v příslušném režimu
03_kontrola funkce tlačítek	Testování funkčnosti zobrazených tlačítek

Zdroj: Autor práce

Do testovacích kroků byly vloženy parametry, které budou nastaveny na hodnotu, která bude relevantní s případem, který je potřeba otestovat. Parametry se v aplikaci ALM definují pomocí znamének, které určují nerovnost („<“ a „>“). Vždy je nutné před a za název parametru vložit 3 tyto znaky. Vkládají se přímo do testovacího kroku. Ukázka vložených parametrů do kroku je vidět na Obrázku 13.

Obrázek 13 Ukázka parametrů vložených do kroku.

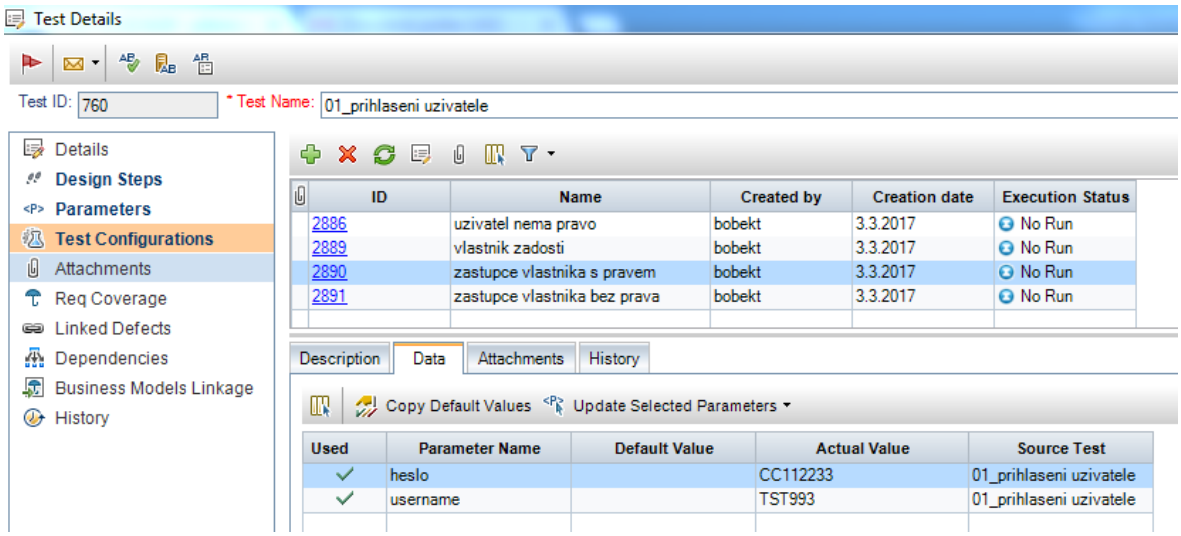


Zdroj: Autor práce

Následně byly vytvořeny konfigurace, které se skládají právě již z výše zmíněných parametrů. Název konfigurace by měl být vždy volen tak, aby bylo na první pohled jasné, jaký případ daná konfigurace ověřuje. Konfigurací pro daný test může být nespočet. Každá

konfigurace obsahuje seznam všech parametrů, které jsou využity v daném testu. Konfigurace se v aplikaci ALM vytvářejí na kartě Test Configurations. Po vytvoření konfigurace a následném jejím zvolením je potřeba definovat hodnoty parametrů. Hodnoty parametrů se definují na kartě Data, vždy je potřeba mít označený řádek zvolené konfigurace, které bude přiřazována hodnota parametrů. Hodnoty parametrů se přiřazují ve sloupci Actual Value. (Viz Obrázek 14)

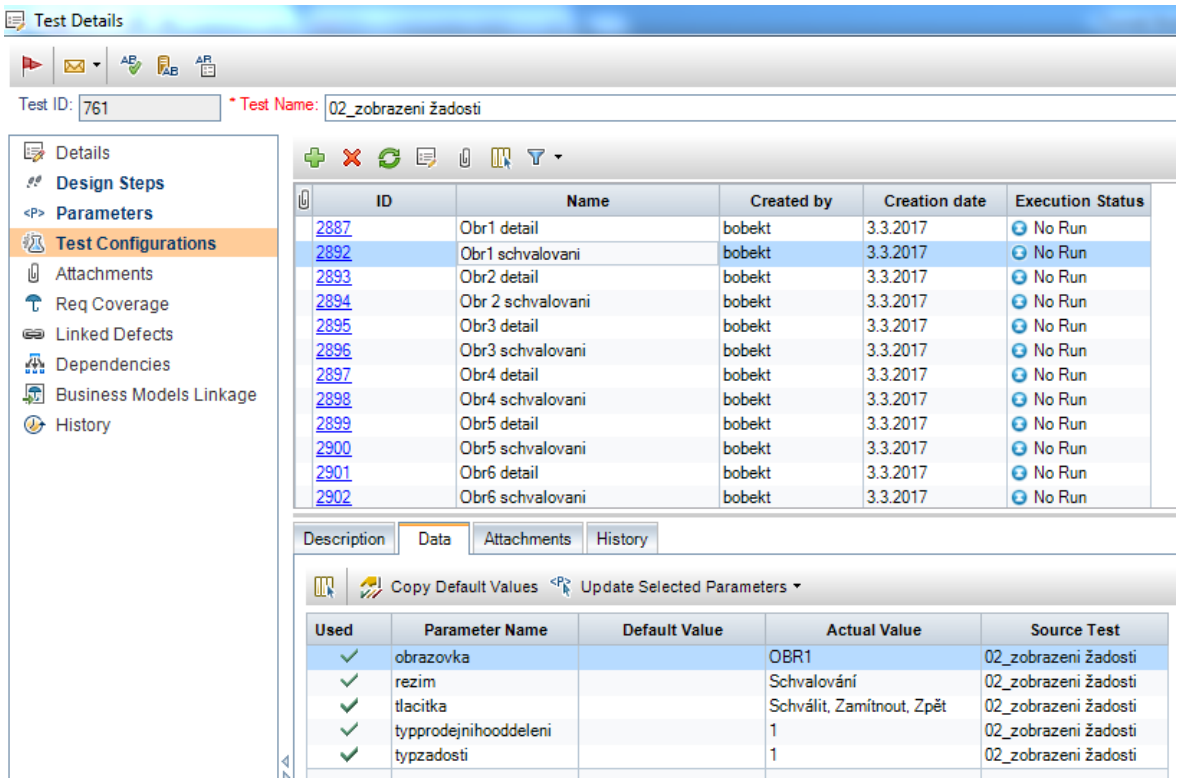
Obrázek 14 Ukázka konfigurací a definovaných parametrů.



Zdroj: Autor práce

Největší počet konfigurací a parametrů bylo potřeba vytvořit u testu *„02_zobrazeni žadosti“*, neboť tento test pokrývá nejobsáhlejší část diagramu. Tato část se zabývá otestováním zobrazení obrazovky a režimu zobrazení v závislosti na parametrech typu: typ žádosti a typ prodejního oddělení. Pro ukázku je přiložen Obrázek 15 s kompletním seznamem všech konfigurací a parametrů použitých v tomto testu.

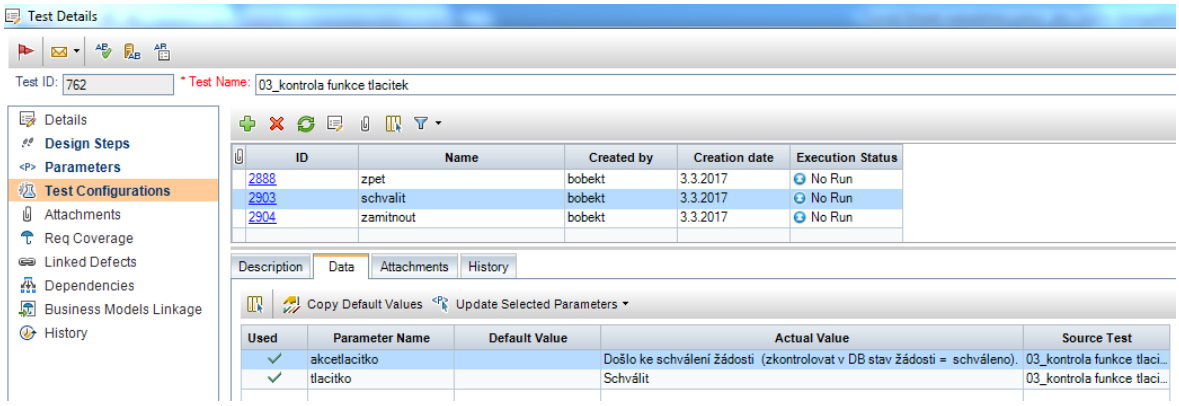
Obrázek 15 Seznam konfigurací a parametrů použitých v testu 02_zobrazeni žadosti.



Zdroj: Autor práce

Pro názornost kompletního řešení dále přikládám Obrázek 16, který zobrazuje seznam konfigurací a parametrů obsažených v posledním testu *03_kontrola funkce tlačítek.* Tento test má za úkol otestovat funkčnost zobrazených tlačítek, zda proběhne zvolená aktivita na základě kliknutí na vybrané tlačítko.

Obrázek 16 Seznam konfigurací a parametrů použitých v testu 03_kontrola funkce tlačítek.



Zdroj: Autor práce

4.4.3 Složení testů do testovacích sad

Pokud jsou testy pro pokrytí diagramu vytvořeny, je potřeba tyto testy podobně jako u procedurálního způsobu připravit pro spuštění. Opět bude využit modul Test Lab, kde

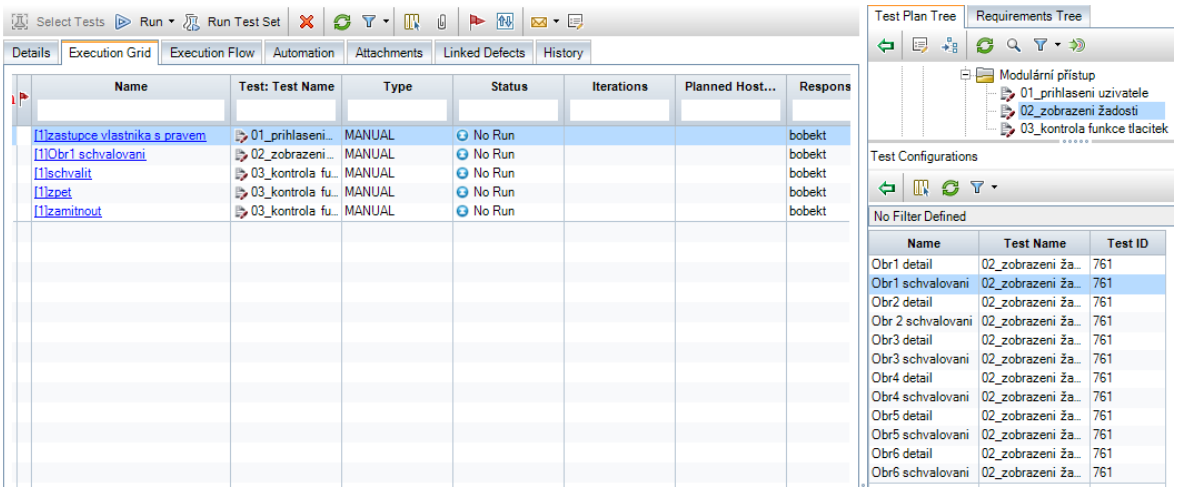
budou vytvořeny testovací sady. Modulární způsob je charakteristický tím, že pro každý průchod větví diagramu je potřeba jedna testovací sada. Na rozdíl od procedurálního způsobu, kde stačí vytvořit pouze jednu testovací sadu, do které se vloží všechny vytvořené testy.

Postup skládání testů do testovacích sad je obdobný jako v případě procedurálního přístupu. Na kartě Test Lab je potřeba stisknout tlačítko *„Select Tests* a v právě části aplikace, kde došlo k zobrazení modulu Test Plan označit test, který chceme do testovací sady vložit. Jelikož testy byly vytvořeny modulárním způsobem, dojde po označení testu k zobrazení k seznamu konfigurací, které byly v daném testu vytvořeny. Pro názornost je v této práci uveden případ, kdy je do testovací sady vložen test, který pokryje větev diagramu, za předpokladu, kdy:

- Přihlášený uživatel je zástupcem vlastníka s právem schvalování
- Typ žádosti = 1
- Typ oddělení = 1

Tato testovací sada nese název *„Režim schvalováníl_typ1_oddeleni1_zastupce vlastnika_schvalit“* a je zobrazena na Obrázku 17.

Obrázek 17 Příklad složení modulů do testovací sady.



Zdroj: Autor práce

Pro kompletní otestování diagramu bylo potřeba vytvořit 24 testovacích sad, přičemž polovina z nich obsahovala 3 scénáře, a druhá polovina 5 scénářů, které je potřeba testerem spustit.

Hlavní nevýhodou tohoto přístupu z pohledu testera, je skutečnost, že pro otestování průchodu jednou větví diagramu je zapotřebí spustit vícero scénářů, kdežto u procedurálního přístupu to byl pouze jeden.

Jako výhodu pro vedení projektu, které je zodpovědné za reportování výsledků testování, je možné jednoznačně určit chybový modul diagramu/aplikace, neboť tyto moduly jsou testerem testovány zvlášť.

4.4.4 Úprava scénáře vytvořeného modulárním přístupem

Jelikož jednotlivé kroky scénáře obsahují definované parametry, není potřeba každý krok scénáře upravovat zvlášť. Stačí pouze přejít do karty Test Configurations daného testu a parametr upravit na nový údaj. Změna se tudíž propíše do každého scénáře, ve kterém je daný parametr obsažen.

5 Výsledky a diskuse

Procedurálním přístupem bylo pro kompletní otestování diagramu potřeba vytvořit 24 testovacích scénářů. Časová náročnost tvorby a přípravy scénářů ke spuštění procedurálním způsobem byla změřena na 50 minut. Následná exekuce scénářů testerovi zabere 8,5 minuty na jeden scénář.

Modulárním způsobem byly vytvořeny 3 scénáře, které dohromady čítaly 19 konfigurací a 9 parametrů. Následně byly složeny do 24 testovacích sad, které dohromady čítaly 96 spustitelných scénářů. Časová náročnost tvorby a složení do testovacích sad činila 1 hodina a 15 minut. Exekuce jedné testovací sady složené ze scénářů napsaných modulárním způsobem byla v průměru 9 minut. Důvodem proč se doba vykonání testu liší, je, že tester musí pro otestování průchodu jedné větve diagramu spustit vícero scénářů (3-5), než tomu tak bylo v případě procedurálního přístupu, kdy tester spustil pouze jeden scénář.

Při úpravě již vytvořených testovacích scénářů byla zjištěna hlavní nevýhoda procedurálního přístupu, neboť je potřeba každý vytvořený scénář upravit zvlášť, kdežto v případě modulárního přístupu, stačí upravit pouze parametr v kartě Data a jeho následná změna je propsána do scénářů, do kterých je vložen. Časová náročnost úpravy všech scénářů v případě procedurálního přístupu činila 30 minut, kdežto u modulárního pouze 5 minut.

Pro účely výpočtu finanční náročnosti na vytvoření testovacích a následné spuštění scénářů byla potřeba zjistit průměrná měsíční mzda pracovníků na pozici Test analytik a Tester. Portál platy.cz uvádí hrubou měsíční mzdu IT testera 32.323 Kč[12] a 41.590 Kč[13] na pozici test analytika. Pro účely výpočtu je potřeba hodnoty přepočítat na mzdu na minutu. (viz Tabulka 12)

Tabulka 12 Výpočet mezd Testera a Test analytika.

Pozice	Na měsíc	Na den[14]	Na hodinu	Na minutu
Tester	32 323 Kč	1470	183,75	3,1
Test Analytik	41 590 Kč	1890	236,25	4,4

Zdroj: Autor práce

[12] Zdroj: http://www.platy.cz/platy/informacni-technologie/it-tester (7. 3. 2017)
[13] Zdroj: http://www.platy.cz/platy/informacni-technologie/it-analytik (7. 3. 2017)
[14] Hodnota byla spočítána na základě výpočtu průměrného počtu pracovních dní v měsíci za rok 2017 (22)

Hodnoty uvedené v Tabulce 10 nejsou skutečnými náklady zaměstnavatele, neboť zde není započtena superhrubá mzda a další náklady. Hodnoty jsou tedy pouze orientační a slouží pouze pro porovnání.

5.1 Příprava dat pro metodu váženého součtu

Jak již nadpis kapitoly napovídá pro porovnání procedurálního a modulárního přístupu bude v této práci využita metoda váženého součtu. Přístupy ke psaní testovacích scénářů budou porovnány na základě následujících parametrů:

- Čas strávený nad analýzou diagramu - zkratka: ČSNAD, jednotka: minuty
- Doba vytváření scénářů- zkratka: DVS, Jednotka: minuty
- Počet scénářů ke spuštění testerem - zkratka: PSKST, jednotka: kusy
- Čas strávený nad úpravou scénářů - zkratka: ČSNÚS, jednotka: minuty
- Doba otestování diagramu- zkratka: DOD, jednotka: minuty
- Finanční náročnost na vytvoření scénářů - zkratka: FNNVS, jednotka: KČ
- Finanční náročnost na otestování - zkratka FNNO, jednotka: KČ

Tabulka 13 Výsledná data pro porovnání přístupů ke psaní testovacích scénářů.

Přístup	ČSNAD	DVS	PSKST	ČSNÚS	DOD	FNNVS[15]	FNNO[15]
Procedurální	2	50	24	30	204	229	631
Modulární	4	75	96	5	216	348	670
Váhy	**0,1**	**0,2**	**0,1**	**0,2**	**0,2**	**0,1**	**0,1**

Zdroj: Autor práce

Již z prvního pohledu na Tabulku 13 je zřejmé, že procedurální přístup dominuje modulární téměř ve všech kritérii, ale pro ověření výsledků bude užita metoda váženého součtu z oblasti vícekriteriální analýzy variant.

5.1.1 Výpočet metody váženého součtu

5.1.1.1 Ideální varianta

H_j	2	50	24	5	204	220	632,4

[15] Hodnoty byly zaokrouhleny na celé číslo

5.1.1.2 Bazální varianta

D_j	4	75	96	30	216	374	669,6

5.1.1.3 Normalizace

Vzorec pro výpočet normalizace = $\frac{X_{ij}-D_j}{H_j-D_j}$

Tabulka 14 Výpočet normalizace.

Procedurální	1	1	1	0	1	1	1
Modulární	0	0	0	1	0	0	0
Váhy	**0,12**	**0,12**	**0,06**	**0,23**	**0,23**	**0,12**	**0,12**

Zdroj: Autor práce

5.1.1.4 Užitek

Užitek jednotlivých přístupů neboli variant je spočítán jako skalární součin normalizované matice a vah pro daná kritéria.

Tabulka 15 Výpočet užitku.

Přístup	**Výpočet užitku**
Procedurální	1*0,12+1*0,12+1*0,06+0*0,23+1*0,23+1*0,12+1*0,12 = **0,77**
Modulární	0*0,12+0*0,12+0*0,06+1*0,23+0*0,23+0*0,12+0*0,12 = **0.23**

Zdroj: autor práce

5.1.2 **Reportování výsledků**

Již z prvního pohledu na tabulku výsledků v kapitole 5.1 bylo zřejmé, že procedurální přístup jasně převyšuje svými kladnými aspekty, které byly zahrnuty do srovnávací analýzy hodnocení, nad modulárním přístupem. Tento fakt potvrdil i následný výpočet pomocí metody váženého součtu, kde nejvyšší užitek přináší právě procedurální přístup s hodnotou užitku 0,77. Modulární přístup přinesl užitek v hodnotě pouze 0,23. Tudíž pro diagram sloužící k účelu této práci, bylo nejvhodnější využít procedurální přístup ke psaní testovacích scénářů.

6 Závěr

Hlavní cíl předkládané BP, porovnat současné postupy při vytváření testovacích scénářů pro ověření kvality SW byl splněn. Na dílčí cíle byly získány tyto odpovědi:

Analyzovat přístupy k hodnocení kvality SW, charakterizovat metody pro testován

Byly analyzovány přístupy k hodnocení kvality SW a charakterizovány metody pro testování. Byly vymezeny a definovány základní pojmy a technologické postupy pro zavedení testovacího procesu softwaru. Dále byly popsány typy přístupů ke psaní testovacích scénářů a uvedeny jejich hlavní výhody a nevýhody a doporučené podmínky pro jejich zavedení.

Prakticky ukázat tvorby testovacích scénářů na základě UML diagramu

V praktické části práce byly dle zadání vytvořeny testovací scénáře (procedurálním a modulárním přístupem) na základě UML diagramu, který byl vytvořen pro účely této práce. Byl popsán a názorně ukázán způsob tvorby testovacích scénářů oběma přístupy. Při vytváření scénářů byly zaznamenávány časové náročnosti tvorby, časové náročnosti na analýzu diagramu, časová náročnost na kompletní otestování diagramu a časové náročnosti na úpravu již vytvořených scénářů.

Zhodnotit pomocí metod vícekriteriálního rozhodování vytvořené scénáře

Dle dalších kritérii jako počet vytvořených scénářů, finanční náročnost na tvorbu scénářů a finanční náročnost na otestování celého diagramu bylo pomocí metody váženého součtu zhodnoceno, který z přístupů ke psaní testovacích scénářů se jeví pro daný diagram jako nejvýhodnější. Práce tedy neříká, který přístup řešení je nejvhodnější pro využití v praxi, nýbrž poskytuje kvantifikované vodítko při rozhodování o užití přístupů ke psaní testovacích scénářů.

Formulovat obecné i specifické závěry

V dnešní době, kdy hraje kvalita finálního produktu velice důležitou roli, chce každý z nás mít jistotu, že za peníze, které zaplatil, dostane kvalitní a spolehlivý produkt, při jehož užívání nevzniknou v budoucnu žádné problémy. Kontrola kvality neboli testování je nedílnou součástí efektivního vývoje softwaru a mělo by na něj být vynaloženo nemalé množství času a prostředků.

Výsledkem vlastní práce bylo zjištěno, že pro UML diagram vytvořený pro účely této práce se jeví využití procedurálního přístupu ke psaní testovacích scénářů jako nejvýhodnější téměř ve všech směrech. Pouze v oblasti modifikace již vytvořených scénářů přináší vyšší efektivitu modulární způsob.

Formulací výsledků práce nebylo záměrem zcela zavrhnout modulární přístup ke psaní testovacích scénářů, přesto že se v tomto případě jeví jako vysoce nevýhodný. V případě velké šířky a hloubky aplikace s velkým počtem případů užití a potřebou testovací scénáře pravidelně udržovat aktuální, se jeho užití jeví jako výhodnější variantou.

V úvodu práce bylo zmíněno, že autor se v oblasti testingu pohybuje několik let, přesto při výzkumu a psaní této práce došel k nejedné nové informaci. Zejména od samotných výsledků výzkumu.

Závěrem by autor rád dodal, že si je vědom, že výzkumem této problematiky na jednom UML diagramu není schopen odhalit veškerou problematiku užití modulárního a procedurálního přístupu ke psaní testovacích scénářů v celé šíři, přesto se snažil přinést co nejkomplexnější zpracování. Tématika přístupu ke psaní testovacích scénářů je tak obsáhlá, že by byla vhodným tématem pro další výzkum.

7 Seznam použitých zdrojů

- Čermák, Miroslav. 2009. „Testování SW“. [online]. [cit. 2.2.2017]. Dostupné z: http://www.cleverandsmart.cz/testovani-sw/
- Foršt, Jiří. Metodika testování. Přednáška 1. 12. 2016.
- Hlava, Tomáš. 2011. „Fáze a úrovně provádění testů“. [online]. [cit. 18. 12. 2016]. Dostupné z: http://testovanisoftwaru.cz/
- Patton, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.
- Platy.cz. 2017. „IT Tester“. [online]. [cit. 7.2.2017]. Dostupné z: http://www.platy.cz/platy/informacni-technologie/it-tester
- Platy.cz. 2017. „IT Analytik“. [online]. [cit. 7.2.2017]. Dostupné z: http://www.platy.cz/platy/informacni-technologie/it-analytik
- Příbrský, Michal. 2012. „Kvantifikovaný přístup k jakosti informačního zabezpečení pro podporu evaluace informačních technologií“. ČZU, Provozně ekonomická fakulta, disertační práce. [online]. [cit. 27. 1. 2017]. Dostupné z: https://www.pef.czu.cz/dl/46198
- Vaníček, Jiří. Information systems quality rating. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2010. ISBN 978-80-213-2062-8.
- Zientková, Jana. 2016. 03 Metodická příručka pro návrh a implementaci testů (moduly Test Plan a Test Lab). Česká spořitelna. [online]. Dostupné z: dokument určený k pracovním účelům – k dostání na vyžádání.