Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Regresní testování

Daniel Jakubský

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Regresní testování" 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 literatury 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 13. 3. 2017 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Janu Tyrychtrovi, Ph.D. za pomoc při této bakalářské práci.

Regresní testování

Souhrn

V bakalářské práci Regresní testování se obecně zabývá testováním softwaru a jeho technikami, konkrétně pak technikou regresního testování softwaru a ukazuje její užitečnost na konkrétních příkladech z praxe.

Práce popisuje analýzu postupu testování, srovnání druhů testování a výběr vhodného testovacího nástroje. Tato teorie slouží jako podklad pro začínající softwarové testery a osoby podílející se na jednotlivých částech softwarového vývoje.

Ve vlastní práci jsou sledovány výsledky a průběh několika testovacích scénářů ve čtyřech kolech regresního testování softwaru, pomocí nástroje HP Quality center. Tyto výsledky jsou dále pozorovány pomocí statistických metod. Ve druhé podkapitole hlavního cíle se zabývá vlivem softwarové chyby na ostatní testy v průběhu testovacích kol.

Z vlastní práce jsou formulovány její výsledky, na jejichž základě je utvořen návrh na zlepšení a zefektivnění testovacích prací.

Stěžejním přínosem práce je vznik určitých doporučení pro Test manažery a Test Analytiky, která vyplívají z výsledků vlastní práce.

Na základě těchto doporučení je dále možné rozhodovat o budoucím plánování testovacích prací, lidských zdrojů a dalších opatřeních napomáhajících snadnému průběhu projektu a jeho zefektivnění.

Klíčová slova: regresní testování, testování softwaru, uvolnění, distribuce, beta testování, kvalitativní testování, testování, chyby, defekty, vývoj, analýza, metody.

Regression testing

Summary

General objective of Bachelor thesis Regress testing is testing software and its technique, specifically with technique of regres testing of software and shows it usefulness on the specific examples in the practice.

This work descibes the analysis of the testing method, comparison of the testing species and selecting appropriate testing tool. This theory serves as a basis for beginning software testers and people participating on individual parts of software development.

In own work are monitoring the results and outcomes over several test scenarios in four rounds of software regression testing, using HP Quality center tool. These results are then observed using statistical methods. In second subchapter of main objective focuses on influence of software error to other tests during the testing rounds.

From own work are formulated its results on which is formed proposals to improve and streamline the testing work.

The most crucial benefit of the thesis is a set of advices and recomendations which should serve as a guideline for the Test Analytics and Test managers.

The future planning, manage the human resources and the further measures whose main purpose is to alleviate the course of the project and make it more effective.

Keywords: regression testing, testing software, release, distribution, beta testing, quality testing, testing,bugs, errors, defects, development, analysis, methods.

Obsah

1 ÚVOD

Dnešní svět je přímo obklopen technologiemi, jako jsou mobilní telefony, tablety, počítače, notebooky a další podobná zařízení. Avšak tato zařízení jsou pouze schránky, které bez aplikací a software neumí zcela nic.

Dnešní společnosti se doslova předhánějí v tom, kdo uvede na trh zajímavější, rychlejší a uživatelsky přívětivější aplikaci.

Než se hotová aplikace (produkt) dostane ke svému uživateli (zákazníkovi), musí urazit velmi dlouhou cestu a projít několika fázemi životního cyklu. Mezi tyto fáze můžeme zařadit procesy testování softwaru, pod které spadá regresní testování softwaru, kterému bude věnována tato práce.

2 CÍL BAKALÁŘSKÉ PRÁCE

Obecným cílem bakalářské práce je poskytnout podklad především pro začínající softwarové testery a pro všechny účastníky softwarového vývoje a obohatit tak jejich znalosti, které mohou sloužit jako základ pro jejich budoucí povolání.

Hlavním cílem práce je představit techniku regresního testování a ukázat jeho užitečnost na konkrétních příkladech z praxe. Z výsledků a výstupů těchto příkladů utvořit jakýsi seznam doporučení pro Test manažery a Test Analytiky.

Dílčí cíle práce jsou:

- srovnání jednotlivých druhů testování,
- srovnání testovacích nástrojů pro zadávání testovacích scénářů,
- analýza postupu testování,
- návrh na zlepšení a zefektivnění testovacích prací.

3 METODIKA BAKALÁŘSKÉ PRÁCE

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů.

Výsledky vlastního řešení jsou realizovány pomocí výstupů z nástroje HP Quality center a pomocí statistických výsledků z nástroje SAS 9.4.

Výstup z nástroje HP Quality center je progresivní graf, který znázorňuje vývoj testů a jejich stav v čase.

Na základě rozboru teoretických poznatků a výsledků vlastního řešení bude formulováno vlastní řešení bakalářské práce, ve kterém bude utvořen návrh na zlepšení a zefektivnění testovacích prací pomocí statistických metod.

Základní statistické metody:

- **Základní charakteristiky procedury Univariate** (Kába, Svatošová, 2012):

Výstupem procedury Univariate v programu SAS 9.4 je výstupní tabulka, obsahující základní popisné charakteristiky proměnné počet, to znamená aritmetický průměr (Mean), směrodatnou odchylku (Std.Deviation), rozptyl (Variance), míry šikmosti (Skewness), špičatosti (Kurtosis), variační koeficient (Coeff.Variation), medián (Median), modus (Mode), variační rozpětí (Range), interkvartilové rozpětí (Interquartile Range).

- **Míry polohy (střední hodnoty)** (Kába, Svatošová, 2012):

Míry polohy udávají polohu středu rozdělení (odtud také název střední hodnoty) a můžeme je rozdělit do dvou skupin. Prvou skupinu tvoří průměry. Ty jsou počítány ze všech hodnot souboru a mohou být vyjádřeny jako průměry prosté – v případě práce s netříděnými údaji či jako průměry vážené – v případě práce s údaji tříděnými, kdy příslušné četnosti představují váhu dané hodnoty. Nejčastěji užívaným je průměr aritmetický, dále pak průměr harmonický a průměr geometrický.

Aritmetický průměr prostý: $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$

Aritmetický průměr vážený: $\bar{x} = \frac{\sum_{i=1}^{k} x_i n_i}{\sum_{i=1}^{k} n_i}$

V praxi užíváme nejčastěji průměr aritmetický. Průměry jsou počítány ze všech hodnot souboru. Vyskytne-li se tedy v souboru nějaká extrémní hodnota či skupina hodnot, je průměr touto hodnotou, resp. hodnotami zkreslen. V takovém případě je vhodnější volit jiné míry polohy.

Do druhé skupiny středních hodnot zařazujeme **medián a modus**.

Medián (značení $\tilde{x}$) definujeme jako prostřední hodnotu řady pozorování, uspořádané podle velikosti. Pokud je počet jednotek *n* zkoumaného statického souboru vyjádřen lichým číslem, bude mediánem hodnota s pořadovým číslem $\frac{n+1}{2}$ (tato hodnota se v daném souboru skutečně vykytuje).

Pokud je rozsah souboru *n* udán sudým číslem, potom tento soubor obsahuje dvě prostřední hodnoty a medián vyjádříme jako průměr těchto dvou hodnot (medián je v tomto případě umělá hodnota, která se v původním souboru nevyskytuje).

Medián se velmi dobře uplatňuje u souborů malého rozsahu, kde by extrémní hodnota značně ovlivnila velikost průměru. Výhodou mediánu je jeho necitlivost k extrémním hodnotám a rovněž to, že k jeho určení nepotřebujeme znát všechny hodnoty znaku v souboru, můžeme ho tedy určit i tehdy, jestliže získané údaje o hodnotách znaku v souboru jsou neúplné.

Další jednoduchou charakteristikou polohy je **modus $\hat{x}$**. Definuje se jako nejčetnější hodnota znaku (je tedy hodnotu nejtypičtější pro daný soubor). Kromě netečnosti k extrémním hodnotám znaku existují i další důvody pro používání modu. V souborech, kde nelze jednotlivé obměny znaku kvantitativně odstupňovat, je modus jedinou možnou charakteristikou polohy. V některých případech je znalost nejtypičtější hodnoty užitečnější než znalost průměrné hodnoty.

- **Míry variability** (Kába, Svatošová, 2012):

Informace o poloze středu, kterou poskytují střední hodnoty, musí být doplněna o informaci o rozložení hodnot kolem tohoto středu či celkově o rozptýlení hodnot daného souboru. Proto ji doplňujeme o charakteristiky variability.

Nejjednodušší charakteristiku variability představuje **výběrové variační rozpětí R**, které je definováno jako rozdíl nejvyšší a nejnižší hodnoty v souboru:

$$R = x_{max} - x_{min}.$$

Pro svou jednoduchost a rychlý výpočet je využíváno zejména při analýzách malých souborů. Jeho nevýhoda spočívá v závislosti pouze na krajních hodnotách, kdy případná extrémní hodnota výrazně ovlivní jeho velikost a navíc o variabilitě znaku u základní masy statistických jednotek nevypovídá vůbec.

Nejčastěji používanými charakteristikami variability jsou **rozptyl a směrodatná odchylka**, které informují o rozptýlenosti jednotlivých hodnot znaku kolem výběrového průměru. **Výběrový rozptyl s^2** je založen na čtvercích odchylek hodnot od aritmetického průměru. $s^2 = \frac{1}{n-1} \times \sum_{i=1}^{n} (x_i - \bar{x})^2$

Prostá forma:

Vážená forma: $s^2 = \frac{\sum_{i=1}^{k} (x_i - x)^2 n_i}{n-1}$.

Rozptyl je součástí mnoha dalších statistických postupů, pro praktické posouzení variability znaku je však vzhledem k tomu, že pracuje se čtverci odchylek (i jednotky sledovaných znaků jsou tedy uváděny jako mocniny), méně vhodný. Mnohem častěji v tomto případě používáme jeho druhou odmocninu, která se nazývá **výběrová směrodatná odchylka:** $s = +\sqrt{s^2}$

Uvedené míry variability představují **míry absolutní**. Chceme-li však porovnávat variabilitu více souborů, které se liší velikostí průměrů, či dokonce jsou každý uváděn v jiných jednotkách, musíme užít **míru relativní**, kterou je **variační koeficient V**:

$$V = \frac{s}{\bar{x}} \times 100\ [\%]$$

- **Kvantilové míry** (Kába, Svatošová, 2012):

Další částí výstupu procedury Means jsou Kvantily, které patří mezi takzvané kvantilové míry.

Kvantily jsou hodnoty, které dělí uspořádaný statistický soubor na určitý počet stejně obsazených částí. Z kvantilových charakteristik se při analýze statistických dat využívají zejména kvartily, které dělí uspořádaný soubor na čtyři stejně obsazené části. První (též nazývaný dolní) kvartil Q1 ($\tilde{x}_{0,25}$) odděluje 25% nejmenších hodnot znaku od zbývajících. Prostřední kvartil (je totožný s již výše zavedeným mediánem) dělí uspořádaný výběr na dvě stejně obsazené části. Třetí, resp. horní kvartil Q3 ($\tilde{x}_{0,75}$) odděluje 75% uspořádaných hodnot znaku od 25% největších hodnot znaku.

Dalšími představiteli kvantilů jsou decily, které dělí uspořádaný soubor na deset stejně obsazených částí a percentily, které dělí datový soubor na sto stejně obsazených částí.

Diference horního a dolního kvartilu se nazývá kvartilové rozpětí.

Polovina tohoto rozpětí se nazývá kvartilová odchylka. Tato charakteristika představuje tzv. robustní alternativu směrodatné odchylky. Její výhodou je, že není ovlivňována extrémně malými, resp. extrémně velkými hodnotami analyzovaného datového souboru.

- **Extrémní pozorování (Kába, Svatošová, 2012):**

V bloku extrémní pozorování je standardně vždy uváděno pět nejnižších (Lowest) a pět nejvyšších (Highest) hodnot. Je třeba upozornit, že se nemusí jednat o extrémní hodnoty ve smyslu definice statistických extrémů.

- **Normální rozdělení (Kába, Svatošová, 2012):**

Hustoty jako pravděpodobnostní modely mohou být velmi různorodé a jejich výběr je teoreticky neomezený. Ukazuje se však, že při popisu reálných procesů (ekonomických, biologických, technických apod.) lze vystačit s relativně malým počtem pravděpodobnostních rozdělení, jež lze aplikovat v překvapivě velkém počtu situací. Dominantní postavení mezi zákony rozdělení náhodných veličin a jejich aplikacích ve statistice má **normální (Gaussovo) rozdělení.** V zásadě lze říci, že je adekvátním pravděpodobnostním modelem takových náhodných veličin, jejichž kolísání je způsobeno sumárním působením velkého počtu nezávislých nebo pouze slabě závislých veličin (vlivů), přičemž příspěvky těchto jednotlivých veličin jsou nepatrné a žádná výrazně nepřevládá nad ostatním (souvisí to s tzv. centrální limitní větou teorie pravděpodobnosti). Uvedeným podmínkám vyhovuje mnoho důležitých veličin, s nimiž se v praxi setkáváme – například chyby měření, výnosy téže plodiny na různých pozemcích, výška sobo stejného pohlaví, testové výsledky ve studiu, životnost výrobků, měření většiny charakteristik živých organismů atd. Hustota pravděpodobnosti normálního rozdělení je dána výrazem: $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \times e^{\frac{(x-\mu)^2}{2\sigma^2}}, \; -\infty < x$, kde $\mu \in (-\infty,\infty)$

a σ^2 jsou parametry tohoto rozdělení, které představují střední hodnotu a rozptyl normálně rozdělené náhodné veličiny. Rozdělení dané uvedenou hustotou bývá

zkráceně označováno $N(\mu, \sigma^2)$. Jestliže $\mu = 0$ a $\sigma^2 = 1$, hovoříme o normovaném normálním rozdělení, jež se označuje N(0,1).

Grafem hustoty normálního rozdělení je tzv. Gaussova křivka. Jedná se o zvonovitou symetrickou křivku, která nabývá maxima v bodě $x = \mu$, při $x \to \pm\infty$ se asymptoticky přibližuje k ose úseček. Pro $x = \mu \pm \sigma$ má Gaussova křivka inflexní body. Změnou střední hodnoty μ při konstantní velikosti rozptylu σ^2 se posunuje Gaussova křivka podél osy x, aniž by se měnil tvar křivky. Mění-li se parametr σ^2 a střední hodnota μ zůstává konstantní, dochází ke změně tvaru normální křivky.

- **Box plot (krabicový diagram)** (Kába, Svatošová, 2012):

Je grafickou prezentací pětičíselného souhrnu (minimum, maximum, dolní kvartil, horní kvartil, medián). Jedná se o diagram, který zobrazuje data ve tvaru obdélníkové krabice a dvou z ní vybíhajících úseček (vousů). Strany krabice odpovídají hornímu a dolnímu kvartilu, úsečka uprostřed krabice pak medián. Vousy znázorňují další hodnoty, které jsou vymezeny intervalem:

$0{,}5 \times IQR \leq |x - \tilde{x}| \leq 1{,}5 \times IQR.$

V určitých případech může některá z úseček chybět, neboť splývá s dolním, resp. horním kvartilem. Jestliže je jedna z úseček zřetelně větší než druhá, je to signálem asymetrického rozložení dat daného souboru.

Hodnoty, které leží mimo interval vymezený vztahem $0{,}5 \times IQR \leq |x - \tilde{x}| \leq 1{,}5 \times IQR$, jsou vzdáleny od dolního, respektive horního kvartilu o více než 1,5 násobek kvartilového rozpětí IQR a představují tzv. odlehlá pozorování a v diagramu bývají vyznačovány jako izolované body.

- **Q-Q graf (kvantilový graf)** (Kába, Svatošová, 2012):

Jednoduchým vizualizačním nástrojem pro orientační posouzení normality je tzv. QQ graf, též nazývaný kvantilový graf. V tomto grafu jsou porovnávány uspořádané hodnoty (podle velikosti) analyzované veličiny s kvantily teoretického normálního rozdělení. Jestliže analyzovaná data neodporují hypotéze o normalitě rozdělení, jsou body v grafu uspořádány přibližně v přímce.

4 LITERÁRNÍ REŠERŠE

V literární rešerši je popsáno zkoumané téma za použití několika zdrojů tak, aby poskytovalo podklad pro výsledky práce a obecně popisovalo dané téma. Tato část je dělena na podkapitoly, které nejprve uvádějí do dané problematiky a dále jsou členěny od vývoje softwaru po uvedení do problematiky testování, až po srovnávání jednotlivých druhů testování a testovacích nástrojů.

4.1 Pojem regresního testování

Na úvod této kapitoly bude osvětleno, co vůbec pojem regrese znamená a jak je využívána v informatice, konkrétně v disciplíně testování aplikací. Testování aplikací se dělí na mnoho odvětví a existuje spousta metod a druhů testování. Mezi ně spadá i pojem regresní testování, které je základním kamenem kvalitní aplikace a u větších systémů by se bez něj rozhodně nedalo obejít.

V pozdějších kapitolách budou uvedeny příklady na problematice testování internetového bankovnictví a testování v oblasti telekomunikací, které na regresním testování celé stojí a je zde snadno pochopitelné.

4.1.1 Regrese

Pojem regrese vychází původně z latinského slova regressus a znamená pohyb zpět, návrat nebo ústup, slovem opačným k tomuto slovu je progres, což znamená pokrok neboli pohyb vpřed. (Otto, 1904)

4.1.2 Regresní testování softwaru

Hlavní a nezbytnou součástí údržby softwarového produktu je aktivita regresního testování. Softwarový produkt je neustále upravován a je na něm ověřováno, zdali se chová podle daných požadavků. (Last, Kandel, Bunke)

Zřejmě nejvíce vystihující pojem regresního testování nalezneme v glosáři ISTQB, což je mezinárodní slovník pojmů, který má za cíl sjednotit komunikaci v testování softwaru a souvisejících oborech. A zní: „Testování dříve otestovaného programu po jeho úpravě, s cílem ověřit, aby v důsledku změn nebyly do upravovaných

částí programu zaneseny nové softwarové chyby nebo nebyly odkryty stávající. Testování je prováděno, zatímco software nebo jeho prostředí je měněno.“ (Black, 2007)

4.2 Proces softwarového vývoje

V této podkapitole bude podrobně popsán vývoj softwaru společně s rolemi, které do něj vstupují.

4.2.1 Role vstupující do vývoje softwaru

Do softwarového vývoje vstupuje řada rolí nebo profesí, které pracují na jednotlivých fázích softwaru. Ne všechny projekty však tyto role využívají, někde jsou tyto role pojmenovány jinak a v některých publikacích jsou ještě dále členěny. V některých případech se prolínají a pracovníci tak zastávají více profesí. Pro naše účely bude postačovat základní rozdělení podle Rona Pattona. Zde jsou uvedeny společně s jejich hlavními úkoly (Patton, 2002, s. 26):

- **Manažeři** (vedoucí) projektu řídí projekt od jeho začátku do konce. Obvykle jsou odpovědní za psaní specifikace produktu, za plánování postupu prací a za provádění podstatných rozhodnutí a kompromisů.
- **Architekti** neboli systémoví inženýři jsou technickými experty produktového týmu. Jsou obvykle velice zkušení, a tudíž mají výbornou kvalifikaci pro návrh celkové architektury systému neboli návrhu softwaru. Velice úzce spolupracují s programátory.
- **Programátoři**, vývojáři neboli kodéři navrhují software, píší jej a odstraňují v něm nalezené chyby. Při tvorbě softwaru úzce spolupracují se softwarovými architekty, s manažery projektu a s testery při opravě nalezených chyb.
- **Testeři** neboli pracovníci zajišťování kvality (Quality Assurance, QA) jsou odpovědní za vyhledávání a oznamování problémů v softwarovém produktu. Úzce spolupracují se všemi členy týmu, a to při vývoji a provádění testů, a také při oznamování nalezených problémů.

4.2.2 Modely životního cyklu vývoje softwaru

Pro vývoj softwaru se používá mnoho různých modelů. Některé z nich jsou na světě už celá desetiletí, zatímco se snad každý týden objevují nové. Některé modely jsou extrémně formální a strukturované, jiné jsou velmi flexibilní. Žádný z nich samozřejmě nebude fungovat pro úplně každý tým, ale vývojovému týmu zpravidla pro vytvoření lepšího produktu pomůže, když se drží některého osvědčeného modelu. Pochopení toho, jaké části vývoje a testování se provádějí v které fázi produktového cyklu, napomáhá vývojovému týmu předvídat některé typy problémů a včas rozpoznat, že problémový návrh nebo potíže v oblasti kvality mohou ohrozit jejich schopnost dodat produkt včas. (Page, Johnston, Rollison, 2009)

Některý software se vyvíjí s přísnou disciplínou pečlivého řemeslníka, některý pod vládou mírně řízeného chaosu, a jiný software je doslova sdrátován a slepen. Zákazník obvykle nakonec pozná, jaký postup se při tvorbě softwarového díla skutečně používal. Proces, podle kterého se vytváří softwarový produkt, a to od jeho prvotního záměru do veřejného uvedení na trh, se nazývá model životního cyklu vývoje softwaru.

Při vývoji softwaru je možné použít celou řadu různých metod, přičemž žádný model se nedá ani u konkrétního projektu označit za jediný správný. (Patton, 2002)
Čtyři nejčastěji používané modely s jejich specifikacemi:

- **Model velkého třesku**

Půvab metody velkého třesku spočívá zejména v její jednoduchosti. Je k ní potřeba jen velmi málo plánování, rozvrhování práce a formálního vývoje - pokud vůbec. Veškeré úsilí se soustředí jen na samotný vývoj softwaru a psaní jeho programového kódu. Pokud vstupní požadavky na produkt nejsou příliš dobře srozumitelné a datum konečného odevzdání je hodně volné, je tento proces opravdu ideální. Současně je ovšem důležité mít velice pružné zákazníky, protože ani oni neví, co nakonec z programátorů vypadne. (Patton, 2002)

Ve většině případů se při modelu velkého třesku žádné nebo téměř žádné formální testování neprovádí. Jestliže vývojový tým k nějakému testování přece jen sklouzne, odehraje se zpravidla těsně před konečným uvedením produktu na trh. (Patton, 2002)

Pro softwarové testery, kteří dostanou za úkol testovat softwarový produkt, vyvíjený v modelu velkého třesku, to znamená snadnou a zároveň obtížnou práci. Software je již téměř hotový, takže máte dokonalou specifikaci - tou je samotný produkt. A protože je prakticky nemožné se v projektu vracet a opravovat věci, které nefungují, nezbývá nic jiného, než nalezené chyby sepsat, aby zákazníci alespoň věděli, do čeho jdou. (Patton, 2002)

Nevýhodou takového testování je, že z pohledu řízení projektu je již produkt připraven k expedici, takže práce testera v podstatě jen zdržuje dodání zákazníkovi. (Patton, 2002)

- **Model „programuj a opravuj“**

„Nikdy není čas udělat to pořádně, ale vždycky je čas to nějak předělat.“ V podstatě přesně o tom je model „programuj a opravuj“. Tým, který se drží tohoto postupu, začíná obvykle hrubou představou výsledného produktu, udělá si nějaký jednoduchý návrh a poté vstoupí do dlouhého, neustále se opakujícího cyklu programování, testování a opravování chyb. (Patton, 2002)

Jako tester projektu vyvíjeného v modelu „programuj a opravuj“ si musíte uvědomit, že budete spolu s programátory pracovat v neustálém cyklu. Prakticky každý den dostanete na stůl novou nebo alespoň aktualizovanou verzi softwaru a budete ji muset vyzkoušet. Provedete tedy potřebné testy, oznámíte nalezené chyby a hned dostanete další verzi softwaru. (Patton, 2002)

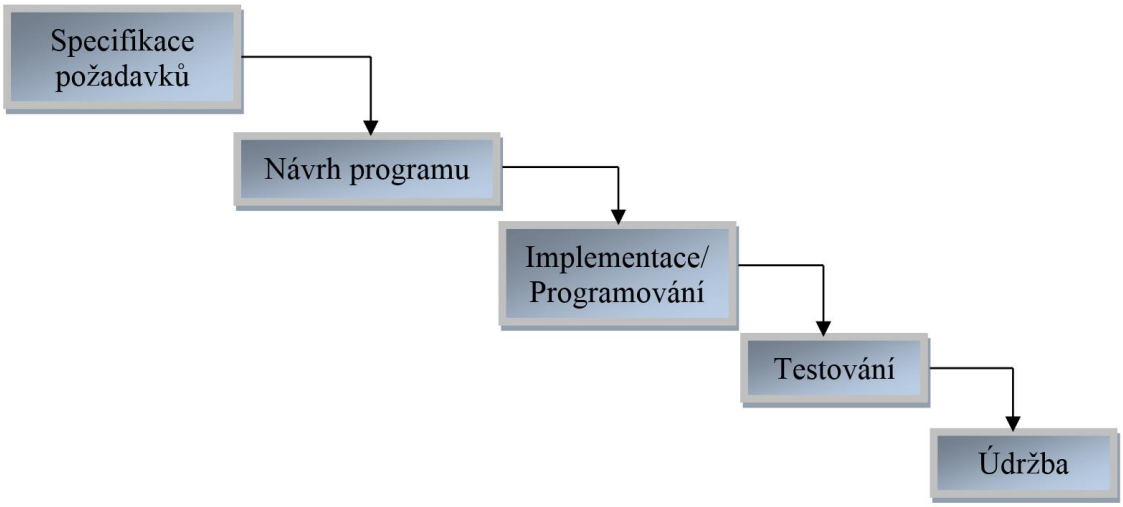
- **Model vodopádu**

Jeden z nejznámějších (a nejvíce zneužívaných) modelů vývoje softwaru je takzvaný „vodopádový“ model, běžně nazývaný Waterfall. Tento přístup spočívá v tom, že konec každé fáze projektu zároveň představuje začátek další fáze. Práce je rozdělena do jednotlivých kroků, které následují v předem určeném pořadí. Provádění prací „přetéká“ z jedné fáze do druhé. (Page, Johnston, Rollison, 2009)

Výhodou tohoto modelu je, že jakmile začínáte kteroukoli fázi, veškerá práce z předchozí fáze je hotová (Page, Johnston, Rollison, 2009), všechno je velice pečlivě a důkladně popsáno a specifikováno. V okamžiku, kdy se software předává testovací skupině, bylo již o každém jeho detailu jasně rozhodnuto, všechno bylo zapsáno a včleněno do softwaru. (Patton, 2002) Návrh nikdy nezačne předtím, než jsou určeny všechny požadavky. Další možnou výhodou je, že vás tento model nutí přemýšlet a

navrhovat co možná nejvíce předtím, než se začne programovat. (Page, Johnston, Rollison, 2009)

Mezi nevýhody tohoto modelu spadá neflexibilnost, neboť neumožňuje některé fáze opakovat, jakmile se například během testování objeví chyba, která ukazuje na nedostatek v návrhu. (Page, Johnston, Rollison, 2009) Testování se provádí pouze na konci vývojového cyklu, a proto může ze začátku vývoje dojít k podstatnému problému, na který se přijde až v okamžiku, kdy se podle rozvrhu prací má již hotový produkt umístit na trh. (Patton, 2002) Tato zjevná nepružnost vedla k mnoha kritikám modelu Waterfall. (Page, Johnston, Rollison, 2009)



Obrázek 1: Vodopádový model cyklu softwaru. Zdroj: (Page, Johnston, Rollison, 2009).

- **Spirálový model**

Používá se poměrně často a ukázalo se, že je to pro vývoj softwaru opravdu velice efektivní postup. (Patton, 2002)

Softwarové týmy mohou spirálový model uplatnit tak, že ze začátku naplánují, navrhnou a vytvoří prototyp nebo značně okleštěnou verzi svého produktu. (Page, Johnston, Rollison, 2009) Odráží vztah úkolů s pohotovým vytvářením prototypů, pracující souběžně s navrhováním a budování aktivit. (Hyperthot, 1997)

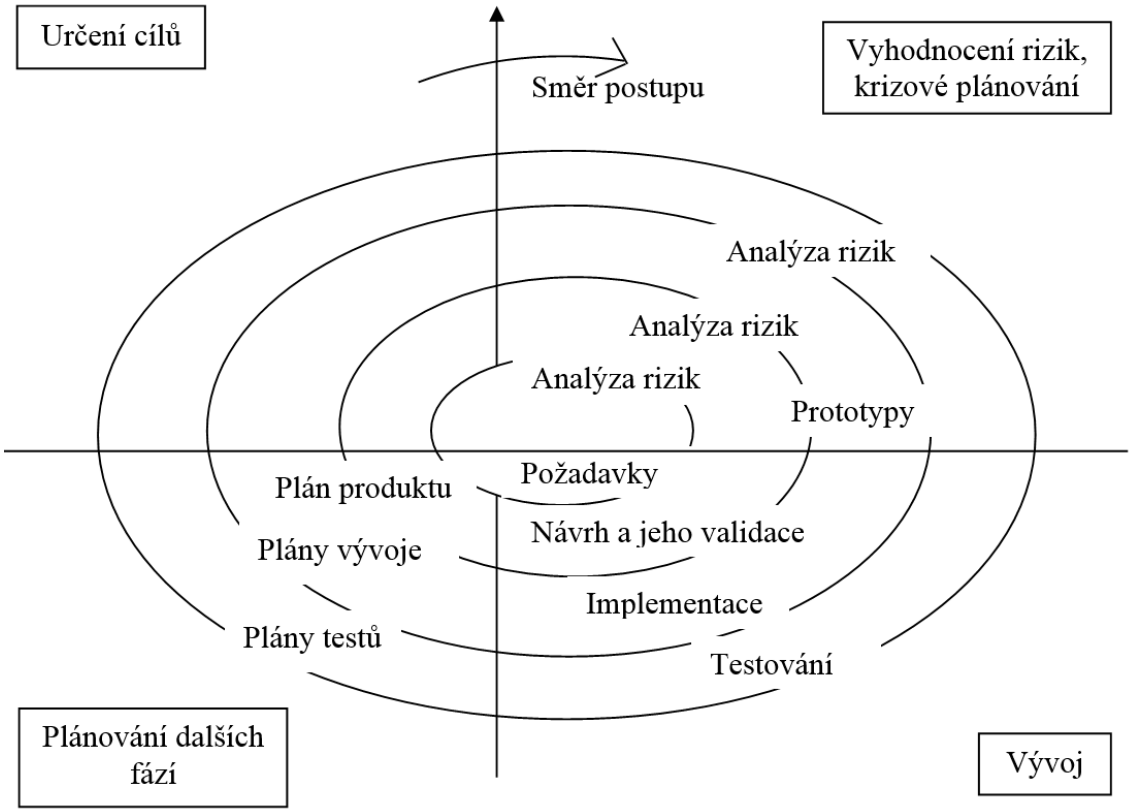
Začneme tedy s malou definicí těch nejdůležitějších funkcí, vyzkoušíme si je, vyžádáme si připomínky od zákazníků a poté přejdeme na další úroveň vývoje. Celý proces opakujeme, dokud se nedostaneme k výslednému produktu. (Patton, 2002) Metoda by měla být plánována vždy systematicky. (Hyperthot, 1997) U každého

průchodu spirálovým modelem se provádějí následující kroky (Page, Johnston, Rollison, 2009):

- **Určení cílů**: identifikace a nastavení konkrétních cílů pro současnou fázi projektu.
- **Vyhodnocení rizik:** identifikace klíčových rizik a cest k jejich omezení, vytvoření krizových plánů. Rizikem může být překročení nákladů nebo problémy se zdroji.
- **Vývoj:** během této fáze se provádí skutečná práce (specifikace požadavků, návrh, vývoj, testování a tak dále).
- **Plánování:** projekt je vyhodnocen a začíná se plánovat další cyklus spirály.

Poté tým získá zpětnou vazbu od zákazníka a analyzuje získaná data pro vyhodnocení rizik a určení úkolů pro další iteraci spirály. Proces pokračuje, dokud není projekt hotov. (Page, Johnston, Rollison, 2009)

Softwarový tester dostane v tomto modelu šanci projekt ovlivnit hned od začátku, protože bude zapojen i do předběžných počátečních fází vývoje. Po celou dobu bude vidět, odkud projekt přišel a kam směřuje. A na konci projektu nebude nikdo testery tlačit, aby veškeré testování prováděli zrovna teď, v posledních minutách. Testovat produkt budou průběžně a neustále, takže v posledním průchodu již pouze ověří, že je všechno skutečně v pořádku. (Patton, 2002)



Obrázek 2: Spirálový model cyklu softwaru. Zdroj: (Page, Johnston, Rollison, 2009).

4.3 Správa a příprava dat

K provádění testů především u rozsáhlejších aplikací (například u internetového bankovnictví, u aplikací pro správu mobilního/internetového tarifu) jsou velmi důležitá kvalitní data. Těmito daty rozumíme například údaje zákazníků, klientů bank, telefonní čísla zákazníků s jejich nastavenými tarify, adresy společností, jejich sídla, atd.

Tyto data pak využíváme k testům pro jednotlivá prostředí. Problematika testovacích prostředí je pro správu dat velmi zásadní. Proto je nutné určit, jaká testovací prostředí budeme pro testy využívat a která pro naše účely nejsou potřebná.

Jako data lze chápat i chyby, které při jednotlivých testech nalezneme. Chyby musíme řádně zpracovat a připravit je pro další osoby podílející se na konkrétním softwaru.

To vše bude popsáno v následujících podkapitolách.

4.3.1 Testovací prostředí

Pokud se disciplínou testování softwaru zabýváme o něco hlouběji, brzy lze přijít na to, že si nelze vystačit s jednoduchým modelem „nejdříve se software vyvine, pak otestuje“. V rámci životního cyklu vývoje softwaru se provádějí různé typy testů, organizované do úrovní testování, které mají různé požadavky na testovací prostředí. (Bureš, Renda, Doležel a kolektiv, 2016)

Podle slovníku *ISTQB* pojmem „testovací prostředí“ je míněno „prostředí“ obsahující hardware, software, simulátory, nástroje a další vybavení potřebné pro vykonání testů. (Bureš, Renda, Doležel a kolektiv, 2016)

Pod hardwarem rozumíme například klientská nebo také koncová zařízení (počítače, tablety…), periferie klientských zařízení (tiskárny, skenery…), serverová řešení, datová úložiště, síťové komunikační prvky. (Bureš, Renda, Doležel a kolektiv, 2016)

Jako příklad softwaru uveďme operační systémy, databáze a aplikační software samotný. (Bureš, Renda, Doležel a kolektiv, 2016)

Typy prostředí a jejich atributy:

Název prostředí	Zkratka	Funkce	Shodnost s produkčním prostředím	Výkonnost	Správa prostředí	Integrovanost
Pískoviště	-	Prototypování, ověřování vývojových a testovacích nástrojů…	Ne	Ne	Oddělení vývoje	Ne
Vývojové (Development)	DEV	Vývojové testy – jednotkové testy, jednotkové integrační testy, testy integrity verze	Ne	Ne	Oddělení vývoje	Ne
Systemtest (Systemtest)	SYS	Vývojové testy integrace, systémové testy – finální testy vývojového týmu	Ano (SW platformy)	Ne	Oddělení provozu	Ano
Integrační (Integration)	INT	Systémové integrační testy a uživatelské akceptační testy	Ano (SW platformy)	Ne	Oddělení provozu	Ano
Předprodukční (Preproduction)	PRE	Zátěžové testy, technické akceptační testy, testy nasazení	Ano (SW i HW platformy)	Ano	Oddělení provozu	Ano
Podpora produkce (Production support)	PRS	Simulování produkčních chyb, testování jejich oprav, testy drobných (konfiguračních) požadavků	Ano (SW platformy)	Ne	Oddělení provozu	Ano
Školící (Education)	EDU	Školení uživatelů	Ne	(Ano)	Oddělení provozu	Ano
Produkce a záloha (Production and backup)	PRO	Produkční prostředí	-	Ano	Oddělení provozu	Ano
	BAC	Záložní prostředí	-	Ano	Oddělení provozu	Ano

Tabulka 1: Typy prostředí. Zdroj: (Bureš, Renda, Doležel a kolektiv, 2016).

Naprostá většina běžného softwaru je vyráběna proto, aby fungovala v produkčním prostředí a poskytovala přidanou hodnotu. Potřebujeme proto produkční prostředí. Máme-li zajistit pro kritické aplikace vysokou dostupnost, je požadováno vybudování záložního prostředí produkce. Existují různé implementace záložního prostředí, ale princip je vždy stejný – toto prostředí převezme zátěž v případě výpadku primárního produkčního prostředí. Tato prostředí mají jednu výhodu – jejich potřebnost pochopí každý. (Bureš, Renda, Doležel a kolektiv, 2016)

Před tím, než začnou aplikaci používat uživatelé, je nutné je někde proškolit. Pokud pomineme školící strategie typ „přečtěte si uživatelskou příručku“ nebo „proklikejte si prezentaci s obrazovkami nového řešení“, máme tu požadavek na školicí prostředí. Školicí prostředí se vyplatí tam, kde je třeba pravidelně školit velké množství lidí a požaduje se praktické osvojení školené aplikace, aby si školený člověk na aplikaci

„sáhl“. I v případě školicího prostředí není problém s pochopením účelu a potřeby tohoto prostředí. (Bureš, Renda, Doležel a kolektiv, 2016)

Aplikací můžeme školit, až když je řádně otestovaná. A tady začíná oblast testovacích prostředí, která skýtá velký prostor pro „akademické“ diskuze ohledně počtu a účelu prostředí. Rozdělme si požadavky na aplikaci na dvě hlavní oblasti - oblast funkcionálních požadavků (například funkcionalita přihlášení do aplikace) a oblast nefunkcionálních požadavků (například maximální doba odezvy přihlašovací stránky, počet současně pracujících uživatelů atd.). Prostředí pro testy nefunkcionálních požadavků (výkonnost, spolehlivost…) nazýváme předprodukční prostředí. Technické testy v předprodukčním prostředí většinou neumožňují současně realizovat jiné typy testů. Například po dobu zátěžových testů nebo po dobu testů instalace aplikace nelze prostředí použít pro jiné aktivity. (Bureš, Renda, Doležel a kolektiv, 2016)

Pokud chceme testovat výkonnost aplikace, je třeba, aby byla aplikace funkční. Pro akceptaci funkčností a integrační testy slouží integrační prostředí. Do integračního prostředí se nasazují hotové funkčnosti, které prošly systémovými testy (finálními testy vývojového týmu). Prostředí pro systémové a systémové integrační testy nazvěme prostředím pro systémové testy (zkráceně systemtest). Toto prostředí je doménou vývojových testerů. Verze se vyvíjí v prostředí zvaném vývojové prostředí. Vývojové prostředí je – jak název napovídá – doménou vývojářů. (Bureš, Renda, Doležel a kolektiv, 2016)

Případné chyby, nalezené v produkčním prostředí, je třeba někde ověřit a otestovat opravy těchto chyb. Často je požadováno nasazení opravy ještě týž den, kdy byla chyba nalezena. K tomuto účelu a ještě pro testy drobných požadavků slouží prostředí pro podporu produkce. (Bureš, Renda, Doležel a kolektiv, 2016)

4.3.2 Správa a příprava testovacích dat

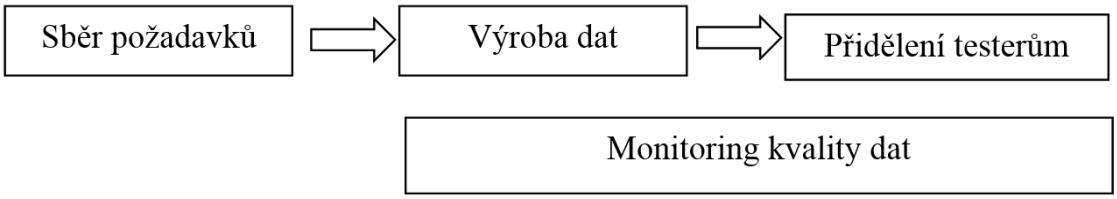
Kvalitní testovací data znamenají úsporu času na uvedení systému do stavu, ve kterém začíná testovací scénář a umožňují zachování stejného stavu systému pro přetestování konkrétního defektu.

Automatizované generování testovacích dat napomáhá testerům usnadnit jejich práci k dosažení softwarové kvality. Za několik posledních let se stává stále více

populární koncept testů řízených vývojem (Test driven development – TDD). Podle TDD by měla být testovací data připravena ještě před začátkem vývoje software. Tento výzkum tvrdí, že by testovací data měla být generována od softwarového návrhu dokumentace, která je vytvořena jako první až po realizaci kódu programu. (Doungsa-Ard, 2011)

Existuje spousta metod a algoritmů automatizovaného generování testovacích dat. Například mohu uvést NSA – Negative selection algorithm nebo ACO – Adapting colony optimization. Tyto metody nebudou v této práci podrobně probírány.

Proces správy testovacích dat je tvořen několika částmi:



Obrázek 3: Proces správy testovacích dat. Zdroj: (Bureš, Renda, Doležel a kolektiv, 2016).

- **Sběr požadavků**

Při tomto procesu se sbírají informace k testovacím datům, které později využijeme k jejich výrobě. Data mohou být velmi specifická, proto je nutné zaznamenat všechny i zdánlivě drobné požadavky, které jsou kladeny na testovací data. Tyto požadavky jsou buď stanoveny v testovacích scénářích, nebo jsou zadávány business testery pro UAT (uživatelské akceptační testy), nebo jsou stanoveny na základě znalostí o produkční verzi systému.

- **Výroba dat**

Podle Miroslava Bureše lze data vyrábět několika způsoby, kterými jsou: ruční typování dat, automatická výroba dat, kopie produkčních dat beze změny dat, částečná kopie produkčních dat, maskování produkčních dat. (Bureš, Renda, Doležel a kolektiv, 2016)

- **Přidělení testerům**

Data jsou testerům přidělována na základě jejich testů, které provádějí. Pro objektivní vykonávání testů je velmi důležité nezasahovat do testovacích dat, která nám nebyla přidělena a neměnit jejich stavy.

- **Monitoring kvality dat**

K tomuto procesu je vhodné využití specializovaného nástroje, v kterém je možné dohlížet na testovací data, jejich přidělení jednotlivým testerům a jejich možné úpravy. Takto sdílené informace jsou velmi důležité pro přehled o stavu dat, který se může měnit a nenarušuje nám tak průchod ve scénářích.

4.3.3 Správa softwarových chyb (defektů)

Než se dostaneme k analýze postupu testování, je potřeba si ujasnit pojmy související s chybami v systému, které jsou zadávány do příslušného nástroje spojeného s jejich správou a umožňující vývojářům chyby pochopit a následně opravit. Každý, kdo se pohybuje v oblasti informačních technologií, si asi pod funkcí softwarového testera představí hledání chyb v programech. Avšak bez pořádné interpretace a identifikace defektu je nalezení chyby bezvýznamné.

Nejprve si definujme, co to takzvaná softwarová chyba je a jaké jsou její příbuzné pojmy.

Podle Pattona o chybě v softwaru hovoříme tehdy, pokud je splněna jedna nebo více z následujících podmínek (Patton, 2002):

- Software nedělá něco, co by podle specifikace produktu dělat měl.
- Software dělá něco, co by podle údajů specifikace produktu dělat neměl.
- Software dělá něco, o čem se produktová specifikace nezmiňuje.
- Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo - podle názoru testera softwaru - jej koncový uživatel nebude považovat za správný.

Správně by pojem chyba v souvislosti s testováním softwaru, ve smyslu nalezené odchylky od očekávaného chování systému, neměl být používán. Vhodnější pojem je anglické slovo „defect“, které se běžně v informatice využívá. Proto je zde chyba uváděna s přívlastkem softwarová.

Podívejme se na pojmy očima praktika podle Zdeňka Grössla (Bureš, Renda, Doležel a kolektiv, 2016):

- Chybu udělá člověk, vývojář či analytik při práci na svém artefaktu (fragmentu kódu, kapitole analýzy, diagramu).
- Defekt (někdy též bug) je pak odchylka aktuálního způsobu fungování softwaru od očekávaného. Defekt je projevem chyby. Je tedy třeba chybu napravit a defekt opravit.
- Selhání (failure) systému může nastat v důsledku výše uvedeného defektu, systém či některá jeho část obvykle zhavaruje.
- Incident je podle definice ISTQB „jakákoliv událost, která vyžaduje prozkoumání.“

4.4 Analýza postupu testování

Obsahem této kapitoly je rozbor postupů a příprava před zahájením testování softwaru a usnadnění prací díky využití testovacích vzorů.

4.4.1 Stanovení testovací strategie

Testovací strategie poskytuje návod pro vytvoření správného návrhu testů a udává testovacímu týmu jasný směr. Dobrá strategie obsahuje týmovou vizi a napomáhá každému členovi týmu určit důležitost jednotlivých aktivit a rozhodnout, kdy a kde použít různé druhy testování. (Page, Johnston, Rollison, 2009)

Podle Pattona (2002) strategie testování popisuje postup, pomocí kterého bude testovací tým daný software testovat, a to jak z celkového pohledu, tak i z pohledu každé jednotlivé fáze.

Strategie popisuje různé typy testů, procesy a metody používané testovacím týmem. Obsahuje vyhodnocení rizik, což týmu umožňuje určit, ve které části produktu lze očekávat nejvíce chyb, nebo které komponenty budou vyžadovat intenzivnější testování. (Page, Johnston, Rollison, 2009)

Proces plánování testů musí každou navrhovanou fázi testování přesně popsat a dát ji na vědomí projektovému týmu. Tento proces často napomáhá zformování celého týmu a vysvětluje mu celkový vývojový model. (Patton, 2002)

4.4.2 Návrh testů a testovacích scénářů

Otázky ohledně testovatelnosti nutí všechny zúčastněné věnovat pozornost tomu, jak bude daný software testován. Tester přitom navíc musí přemýšlet o návrhu testovacích scénářů. Proces návrhu testů je často stejně důležitý jako návrh softwaru pro koncové uživatele. Specifikace návrhu testů (test design specification - TDS) je použitelná pro manuální i automatizované testy. (Page, Johnston, Rollison, 2009)

Jelikož TDS popisuje jak záměr, tak způsob testování, stává se nedílnou součástí testovacího procesu během celého životního cyklu produktu, zvláště pak po jeho vydání, kdy starost o podporu produktu převezme tým údržby. (Page, Johnston, Rollison, 2009)

Podle normy ANSI/IEEE 829 specifikace návrhu testů upřesňuje postup testů (definovaný v plánu testů) a identifikuje jednotlivé funkce, popsané v návrhu, a určuje s nimi spojené testy. Dále identifikuje testové případy a testové procedury (pokud jsou nějaké), které jsou pro zajištění testů potřeba, a stanovuje kritéria pro splnění a nesplnění testů. (Patton, 2002)

Úkolem specifikace návrhu testů je tedy zorganizovat (uspořádat) a popsat testování, které se má provést nad konkrétními funkcemi. Nedozvíme se zde ale podrobně rozepsané případy či kroky, které je třeba při provádění testů provést. Tento úkol rozpracovává následující téma, převzaté z normy ANSI/IEEE 829, které by tudíž mělo být součástí každé vytvořené specifikace návrhu testů. (Patton, 2002)

4.4.3 Používání testovacích vzorů

Testovací vzory řeší často se opakující problémy a nabízejí vodítka a strategie, využitelná testery při návrhu testů. Některé z těchto vzorů popisují postupy pro strukturované testování, jiné představují heuristiky, další jsou kombinací různých myšlenek nebo něčím úplně jiným. Testovací vzory jsou důležité proto, že je testeři mohou využít pro objasnění celkového záměru testu a mohou srozumitelným a jednoznačným způsobem popisovat techniky návrhu testů. (Page, Johnston, Rollison, 2009)

Běžný formát pro popis testovacích vzorů využívá šablonu. Robert Binder v knize Object-Oriented Systems: Models, Patterns, and Tools popisuje šablony

zahrnující deset atributů. (Page, Johnston, Rollison, 2009) Podle Page a spol., lze využít zjednodušenou šablonu, zahrnující následujících osm atributů (Page, Johnston, Rollison, 2009):

- **Jméno** - dejte šabloně snadno zapamatovatelné jméno.
- **Problém** - napište jednu větu popisující problém, který tento vzor řeší.
- **Analýza** - Popište oblast problému (v jednom odstavci) a vysvětlete, proč je lepší použít tuto techniku, než se chaoticky šťourat v kódu
- **Návrh** - popište, jak se tento vzor používá (jak se z návrhu testů vyrobí testovací případy).
- **Věštba** - vysvětlete očekávané výsledky (může být zahrnuto v sekci Návrh).
- **Příklady** - sepište příklady toho, jak použití tohoto vzoru nachází chyby.
- **Nástrahy a omezení** - vysvětlete, za jakých okolností a v jakých situacích by tento vzor neměl být používán.
- **Související vzory** - uveďte seznam souvisejících vzorů (jsou-li takové vzory známy).

Ovšem ne každý tester je zastáncem testovacích vzorů. Autor knihy Testování softwaru Ron Patton je jejich velkým odpůrcem.

Problém tohoto přístupu spočívá v tom, že klade příliš velký důraz na výsledný dokument, nikoli na samotný proces plánování. Je známo, že vedoucí testovacích týmů a manažeři rozsáhlých softwarových projektů si s oblibou vezmou elektronickou kopii nějaké takové šablony plánu testů nebo dokonce existující plán, několik hodin v něm něco „mastí“ - kopírují, přepisují, vkládají, hledají a nahrazují, až z nich takto vypadne „jedinečný“ plán testů pro jejich konkrétní projekt. (Patton, 2002)

4.5 Členění regresních chyb a regresních testů

V této podkapitole bude popsáno rozdělení regresních chyb softwaru, v praxi běžně nazývaných jako defekty, a dále regresní testy z hlediska změn a oblasti dopadu.

4.5.1 Typy regresních chyb a jejich příčiny

Podle Miroslava Rendy se v praxi setkáte se třemi typy regresních chyb softwaru (Bureš, Renda, Doležel a kolektiv, 2016):

- **Lokální defekty**

Změna v kódu konkrétního modulu zanáší do tohoto modulu nový defekt. Například vývojář přidává do formuláře pro zadání platebního příkazu nové textové políčko „Účel platby“, které se má zobrazit při zadané částce vyšší než 300 000 Kč (cílem je implementace nařízení regulátora proti praní špinavých peněz). V původní verzi formuláře byla validační kontrola, která ošetřovala, že formulář neakceptuje nečíselný vstup. S novým kódem byla vývojářem nechtěně změněna také původní validační kontrola a formulář po odeslání tisícové částky včetně mezery zobrazuje chybu.

- **Vzdálené defekty**

Typicky integrační defekty, kde změna v jedné komponentě způsobuje nefunkčnost v jiné. Příkladem může být situace, kdy se v rámci změnového projektu změní rozhraní volané služby a neošetří se, jaké všechny komponenty tuto službu volají. Důsledkem může být nesprávné chování formuláře, který změněnou službu volá, a přitom v něm nebylo nic modifikováno.

- **Odkryté defekty**

Změna v jedné komponentě způsobí, že se projeví defekt, který už dříve v komponentě byl, ale nezpůsobil selhání. Například vývojář dopředu připravil metodu třídy, která se ve staré verzi systému nepoužívala. Tato metoda obsahovala defekt, který ovšem nebyl ve staré verzi nikdy aktivován. Nová verze systému ale tuto metodu již požívat začala a defekt se začal projevovat.

4.5.2 Členění regresních testů z hlediska změn a oblasti dopadu

Z hlediska změn softwarového produktu podle Miroslava Rendy, je možné rozdělit testy na tyto tři kategorie (Bureš, Renda, Doležel a kolektiv, 2016):

- **Testy změn**

Testy ověřující, že došlo ke změnám popsaným v dokumentaci (testovací báze). Tyto testy ověřují detailně implementaci dílčích požadavků. Testy změn testují poměrně úzkou množinu funkcionalit či jiných charakteristik (např. výkonnost), jdou však hodně do hloubky.

- **Regresní testy oblastí dopadu**

Jde o regresní testy v oblastech, kde docházelo k změnám dílčích funkcionalit, nicméně funkcionalita testovaná konkrétním testovacím scénářem se neměnila. Testy mají mnohem širší záběr co do počtu ověřovaných funkcionalit, ale nejdou tolik do detailu jako testy změn (například netestují takové množství kombinací vstupů).

- **Regresní testy neměnných částí mimo oblasti dopadu**

Jde o testy oblastí, v nichž nedocházelo k žádným (vědomým) změnám. Tento typ testů je co do míry detailu nejpovrchnější, ale pokrývá největší šířku funkcionalit.

4.6 Užití regresních testů

S pojmem regresního testování jsou spojeny následující 3 podoblasti testování softwaru.

4.6.1 Regresní testování po nalezení chyby

Jakmile je chyba opravena, stává se z testu regresní test – dokáže tedy ověřit, zda nové změny v kódu nezpůsobily nefunkčnost dříve dobře fungující vlastnosti. (Page, Johnston, Rollison, 2009). V knize efektivní testování softwaru je tomuto druhu testování přiřazován název konfirmační testování, často nazývané také přetestování nebo retestování a je o opětovném spuštění testovacích případů, které v předchozím spuštění skončily s chybou. Jeho cílem je potvrdit, že opravné aktivity „zabraly“. (Bureš, Renda, Doležel a kolektiv 2016)

4.6.2 Regresní testování uživatelského rozhraní

Pojem testování GUI je definován v glosáři ISTQB jako „Testování prováděno interakcí (vzájemné působení, ovlivňování a směřování k cíli skupiny) se softwarem přes testy grafického uživatelského rozhraní. (Black, 2007)

Na trhu existuje řada nástrojů, které testerům umožňují v rámci automatizovaných testů pracovat s uživatelským rozhraním. Řada nástrojů navíc poskytuje funkci „nahrávání“, takže autor testu může jednoduše zaznamenat průběh manuálního testu a následně jej „přehrát“ jako automatizovaný test. Tento způsob

testování má řadu kritiků, neboť nahrané testy nejsou odolné vůči drobným změnám uživatelského rozhraní. Tento problém se týká testování uživatelského rozhraní obecně, a navíc není vždy snadné automatizovat určité ovládací prvky rozhraní. (Page, Johnston, Rollison, 2009)

Softwarové užití grafického uživatelského rozhraní (GUI) může významně zvýšit cenu regresního testování, protože GUI je upravováno a retestováno velmi často a speciálně pak GUI, kde je využíváno technik automatického regresního testování. (Last, Kandel, Bunke, 2004)

4.6.3 Regrese automatizovaných testů

Pokud vytváříme automatizovaný test, vlastně již testujeme. Pokud je v systému defekt, přijdeme na něj už při vytváření testu. Hlavní oblastí, kde jsou automatizované testy účinné, je proto kontrola regrese – ověřujeme, že se v dané oblasti, která již byla bez defektů, nějaký defekt znovu neobjevil. (Bureš, Renda, Doležel a kolektiv, 2016)

Pokud existuje druh testů, u kterých se regrese vyplatí nejvíce, pak je to rozhodně u automatizovaných testů. Takovéto testy můžeme provádět kdykoliv a po jakékoliv drobné změně bez většího zvýšení nákladů. Automatizované testy jsou velmi rychle provedeny a mohou tak být provedeny téměř kdykoliv. A snadno při nich zjistíme, jak nás ovlivnila implementace nového prvku nebo funkčnosti.

Řada technik používaných při automatizaci umožňuje měnit data používaná při testování nebo měnit průchody kódem ověřované během každého běhu, což umožňuje nacházet chyby v průběhu celé životnosti testu. V případě produktů s dlouhou životností je rostoucí sada regresních testů výhodou - vzhledem k vysoké složitosti testovaného softwaru je velmi přínosné mít k dispozici velké množství testů primárně zaměřených na ověřování dříve testovaných funkcí. (Page, Johnston, Rollison, 2009)

Téma automatizovaného testování bude ještě dále probíráno v kapitole 4.8.3 Manuální vs. automatizované testování.

4.7 Problematika regresního testování v jednotlivých odvětvích

Regresní testování softwaru je využíváno především u velkých podniků a společností, u kterých je potřeba zajistit maximální možnou správnost a bezchybnost. Zde jsou uvedeny 2 oblasti, u kterých by rozhodně nemělo chybět.

4.7.1 Regresní testování v bankovnictví

Mezi nejvýznamnější faktory patří dopady produktových rizik. Testování elektronického bankovnictví, s obrovským reputačním rizikem, bude mít diametrálně odlišnou šířku testovaných funkcionalit i míru detailu testů, než regresní test chatovací aplikace pro server internetové seznamky. A nemusíme uvažovat jen o reputačních rizicích, která se obtížně vyčíslují. (Bureš, Renda, Doležel a kolektiv, 2016)

4.7.2 Regresní testování v telekomunikacích

V oblasti telekomunikací může docházet k takzvaným svolávacím akcím, které mají za úkol stáhnout výrobek z prodeje, například díky softwarové vadě. V minulosti k tomuto stažení došlo například u vybuchující baterie mobilního telefonu jedné Korejské značky, která musela zařízení s vadou v softwaru stáhnout z prodeje a měnit přístroje za nové. V takovém případě firma ztrácí dobrou reputaci a náklady šplhají hodně vysoko. Firma se dokonce rozhodla, že celou tuto řadu telefonů úplně stáhne z prodeje, aby nedocházelo k dalšímu poškozování jejího dobrého jména. A dokonce díky této aféře byla v USA a Číně zpřísněna pravidla pro přesun mobilních telefonů v letadle, aby nedocházelo k výbuchům na palubě. Toto vše může mít za následek neodhalená softwarová chyba.

4.8 Srovnání jednotlivých druhů testování

V této podkapitole bude uvedeno srovnání několika druhů testovacích metod a postupů, společně s konkrétními příklady.

4.8.1 Testování černé skříňky vs. testování bílé skříňky

Při použití testů černé skříňky není k dispozici přístup k programovému kódu, (Testování softwaru, 2011) tester ví jen to, co má předložený software dělat - dovnitř skříňky se podívat nemůže a neví tedy, jak pracuje uvnitř, (Patton, 2002) její obsah (zdrojový kód) není zvenčí viditelný. Neznáme tedy, jak přesně systém pracuje s daty. Můžeme pouze sledovat, jaký výsledek získáme po vložení vstupních dat. (Testování softwaru, 2011) Jestliže napíše nějaký údaj na vstupu, dostane určitý odpovídající výsledek. (Patton, 2002)

U testování bílé skříňky (kterému se někdy říká také testování průhledné skříňky) má oproti tomu softwarový tester přístup ke zdrojovému kódu, (Patton, 2002) známe tak vnitřní strukturu software (Testování softwaru, 2011) a jeho zkoumání mu může pomoci při testování - vidí tedy „dovnitř“ skříňky, (Patton, 2002) lépe pak můžeme otestovat pokud možno všechny průchody zdrojovým kódem, zadání neočekávaných vstupních hodnot a další testy (Testování softwaru, 2011) a může odhadnout, jestli určitá čísla povedou častěji nebo méně často ke zkáze a podle těchto informací může lépe přizpůsobit další testování. (Patton, 2002)

4.8.2 Statické vs. dynamické testování

Při statickém testování se testuje něco, co neběží, (Patton, 2002) testy nevyžadují běh software (Testování softwaru, 2011) - to znamená, že zkoumaný objekt pouze prohlížíme a kontrolujeme (revidujeme). (Patton, 2002) Využívají se zejména v raných fázích životního cyklu software, kdy ještě není vytvořen funkční prototyp aplikace. Lze je použít ještě před začátkem psaní kódu na kontrolu specifikace požadavků a analýzu zdrojového kódu. (Testování softwaru, 2011)

Čím dříve začínáme testovat (ověřovat) naše myšlenky, které jsou na začátku projektu ve formě dokumentace a teprve se (v pozdějších fázích projektu) změní ve výstupy projektu, tím dříve můžeme najít problémy a chyby. Pokud bychom je našli až ve vyvinutém softwaru, stála by nás jejich oprava mnohem větší úsilí, než bude stát oprava textového dokumentu. Pokud opravíme chybu v kódu, aniž bychom ho museli spustit, ušetříme jednak strojový čas a jednak čas při následném vyhodnocování výsledků testů. To však není hlavním přínosem statického testování kódu. Přínosem je

kód, který se dá snáze testovat a v případě potřeby rozšiřovat nebo udržovat. Přínos statického testování kódu tedy spočívá v úspoře času, který musíme kódu celkově věnovat. (Bureš, Renda, Doležel a kolektiv, 2016)

S veličinou času souvisí i náklady. Pokud ušetříme čas vynaložený na testování, ušetříme i náklady, které by byly v budoucnu vynaloženy na testování, a to jednak z hlediska času stráveného na testování produktu a jednak také na budoucí opravy defektů, které by vznikly v důsledku špatně otestované dokumentace, v horším případě neotestované dokumentace nebo zdrojového kódu.

Dynamické testování – software spustíme a pracujeme s ním. (Patton, 2002) Testy proto potřebují alespoň spustitelný prototyp softwaru. Používají se v pozdějších fázích vývoje a jsou zaměřeny na provoz softwaru. (Testování softwaru, 2011)

Nejlepší analogií, na které si tyto pojmy můžeme představit, je kontrola ojetého automobilu při koupi. Zkoušíme „zabrat“ za kola, jestli v nich nejsou vůle, podíváme se na stav laku karoserie a otevřeme kapotu – to jsou postupy statického testování. Potom auto nastartujeme, poslouchámе zvuk motoru a absolvujeme zkušební jízdu po normální silnici – to jsou techniky dynamických testů. (Patton, 2002)

4.8.3 Manuální vs. automatické testování

Běžný laik si pod prací testera nejspíše vybaví manuální testování. Na rozdíl od automatizovaného testování zde testuje člověk, který může odhalit některé chyby, které stroj odhalit nedokáže. Takovéto testování je u menších projektů zcela běžné a automatizace se zde nevyplatí kvůli vyšším nákladům. Proto je nutné pečlivě analyzovat, zdali se vyplatí automatizovat nebo ne.

Automatizované testování je z hlediska pracnosti jednotlivého testovacího scénáře bráno za podstatně méně náročné, ale naopak velmi náročné z hlediska na jeho přípravu. Samotný proces automatizovaného testování je prováděn bez použití lidského faktoru s pomocí specializovaného nástroje. Velmi vhodná je pro zdlouhavé a časově náročné procesy, které jsou pro člověka příliš monotónní a lze je snadno automatizovat. Takováto monotónní práce může být pro člověka velmi náročná na udržení pozornosti po celou dobu testování, a proto v ní nemusí nalézt všechny chyby a práce je tak

vykonávána nepřesně. Pokud je proces automatizace správně navržen, pak jsou s velikou pravděpodobností nalezeny všechny jeho chyby a je tak zajištěna kontrola kvality.

Často je o automatizaci rozhodováno na základě počtu spuštění testu. Pokud test bude prováděn jen jednou, nevyplatí se jej automatizovat. Proces automatizace je ve většině případů časově náročný a než se do něj začneme pouštět, měli bychom zvážit, zdali se z hlediska nákladů vyplatí.

Někteří experti automatizované testy považují za hloupou a bezcitnou náhražku skutečného testování, kterého je schopen pouze lidský mozek. Jiní považují za nedostatečné jakékoli testování, jež není plně automatizováno. V praxi ovšem hodnota automatizace závisí na daném kontextu. Někdy je smysluplné automatizovat všechny testy, jindy je lepší neautomatizovat vůbec. Některé chyby nelze odhalit bez toho, aby někdo pozorně sledoval obrazovku s běžící aplikací. Chyby popisované slovy „Tohle je vážně divné – když zavřu tohle dialogové okno, celá obrazovka zabliká“ nebo „Ukazatel myši se pohybuje trhaně“ dokáže člověk odhalit mnohem snáze než počítač. Pro odhalování jiných typů chyb je ovšem automatizace velmi efektivní. (Page, Johnston, Rollison, 2009)

4.9 Srovnání testovacích nástrojů

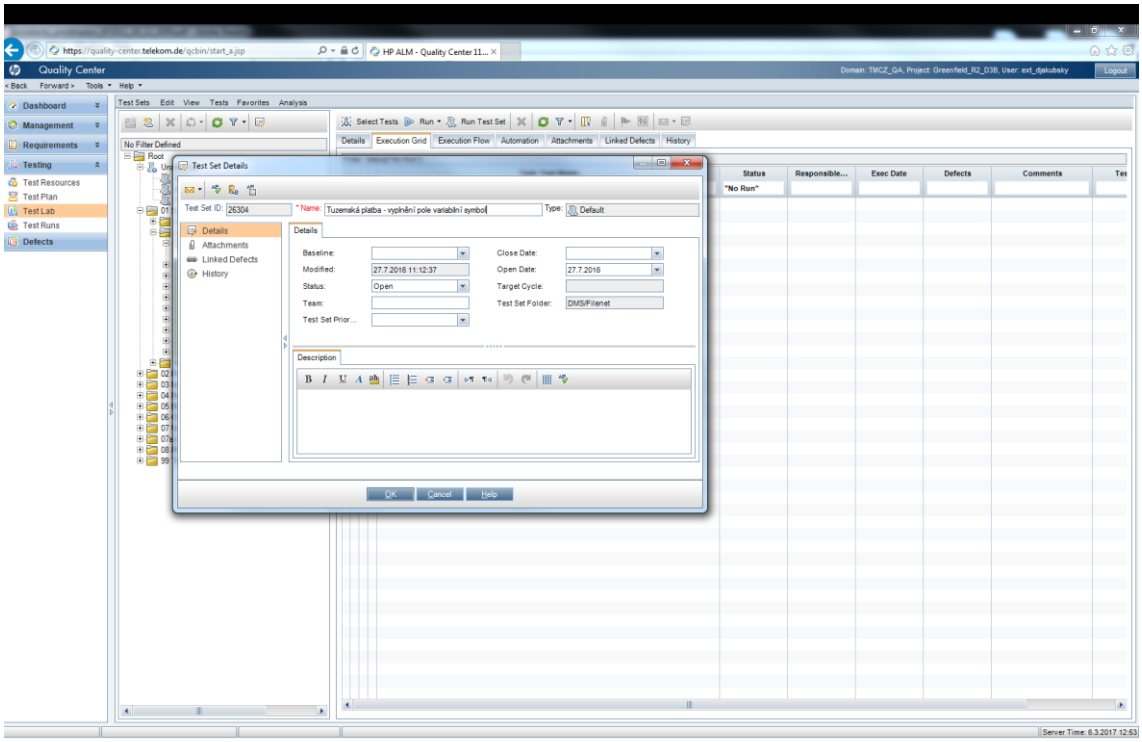
Při provádění testovacích scénářů je potřeba zapisovat jejich výsledky a průběh. K těmto účelům jsou využívány testovací nástroje, které nám umožní snadnou orientaci v testovacích sadách, jednotlivých testech, testovacích scénářích a defektech. Některé nástroje umožňují i podrobné sledování statistik a další nadstavbové funkce, které pomáhají při testování softwaru.

4.9.1 Nástroj HP Quality center

HP Quality Center (QC), je komerční testovací nástroj od společnosti HP, podporující různé faze softwarového vývoje životního cyklu. Tento nástroj je k dispozici jako software, který je nabízen uživatelům přes internetový prohlížeč. (Tutorialspoint, 2017)

Uvnitř nástroje je k dispozici několik funkcí, mezi které patří například: pohled do analýzy, testovací plán, testovací laboratoř, správa defektů a další.

Na následujícím obrázku (Obrázek 4), je konkrétně pohled do testovací laboratoře, ve které je spuštěn detail testovací sady.



Obrázek 5: Testovací nástroj HP Quality center. Zdroj: (vlastní zpracování).

Výhody:

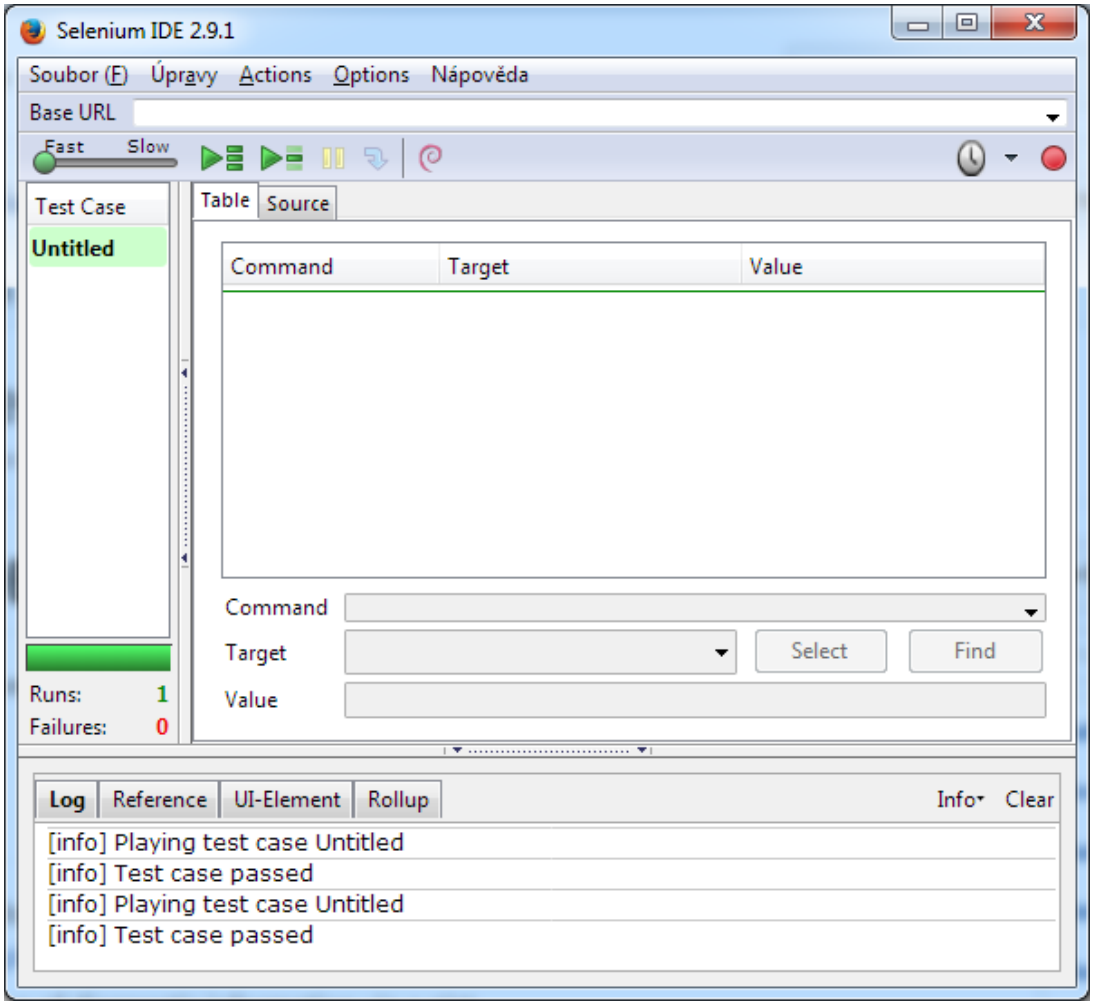
- Široké využití napříč všemi fázemi vývoje.
- Snadná interpretace výsledků testů, jejich grafické znázornění a export.

Nevýhody:

- K dispozici je pouze placená verze.
- Spustitelný pouze v některých prohlížečích.

4.9.2 Nástroj Selenium

Nástroj Selenium je jeden z nejznámějších testovacích nástrojů, který je hojně využíván. Jedná se o volně stažitelný software dostupný všem testerům a vývojářům, kterým umožňuje vytvářet funkční testy pro řízení internetových prohlížečů. Může být používán k zaznamenávání pracovních postupů, které vývojářům mohou zachránit budoucí návraty ke kódu. Selenium může pracovat v kterémkoliv prohlížeči, který podporuje jazyk JavaSkript. (Burns, 2012)



Obrázek 6: Testovací nástroj Selenium IDE. Zdroj: (vlastní zpracování).

Samotný nástroj se skládá z několika komponent (Testování softwaru, 2011):

- Selenium IDE – plugin do prohlížeče Firefox. Jde v podstatě o „nahrávač“ kroků, které ve výsledku tvoří samotné testy. Selenium IDE je nástroj napsaný v jazyku JavaScript. Testy lze pochopitelně psát i bez tohoto rozšíření Firefoxu. Pohlížejme na něj tedy jako na nástroj, který nám může ulehčit práci.
- Selenium RC - neboli Selenium server umožňuje spouštění selenium testů vytvořených v programovacích jazycích PHP, Java, Ruby, Python, Perl nebo .NET. Testy lze spouštět z lokální stanice. Nemusí tedy být na serveru. Selenium server je nyní v Selenium 2 v podstatě jeden java soubor.

- Selenium Grid – Je nástroj umožňující paralelní spouštění Selenium testů. Testy je možné spouštět na jednom stroji (až 15 paralelních testů) nebo na serverových farmách. Dále Selenium RC rozšiřuje možnosti distribuce testů na více serverů. Spouštění testů v prohlížeči není optimální, protože zbytečně zatěžuje stroj, na kterém běží a také server.

Výhody:

- Testy lze spouštět ve většině webových prohlížečích a na mnoha platformách.
- Nástroj je volně k dispozici zdarma.

Nevýhody:

- Nelze využít k testování desktopových aplikací.

4.10 Vliv Regresních testů na rozpočet

Regresní testování je prováděno k zajištění, aby nebyla do programu zanesena nová chyba, a je prováděno mnohokrát za jeho životnost. Dále má hluboký vliv na celkový rozpočet.

Je třeba objektivně určit priority a pořadí testovacích scénářů k dosažení určité objektivity. Obvyklý cíl určení priorit je nalezení softwarové chyby co možná nejdříve během procesu testování. Byla vytvořena spousta technik k seřazení testovacích scénářů podle priorit k dosažení objektivity. Avšak spousta těchto technik byla vytvořena nezávisle jedna na druhé a tak mají mnoho společného. (Malishevsky, 2016)

Nejvhodnější určení priorit testů bude zřejmě kombinace několika technik a algoritmů. Obecné určení priorit dobře popisuje článek Alexeye Malishevskyho, který kombinuje několik struktur. Takovéto setřídění testovacích scénářů nám může značně ovlivnit rozpočet celého projektu a jeho náklady na testování regrese.

5 Výsledky

Kapitola výsledky je rozdělena do dvou částí, na základě kterých je formulováno jejich zhodnocení. Obě části obsahují získaná data v průběhu regresního testování. První část 5.1 Znázornění regresních testů v průběhu testovacích kol je zpracována více dopodrobna a její data jsou statisticky zpracována. Druhá část 5.2 Vliv softwarové chyby na ostatní testy v průběhu testovacích kol, zabývající se vlivem softwarové chyby na ostatní testy, v průběhu testovacích kol, se více zabývá samotným procesem testování pomocí testovacích scénářů a některé vlastnosti jsou zde více názorné.

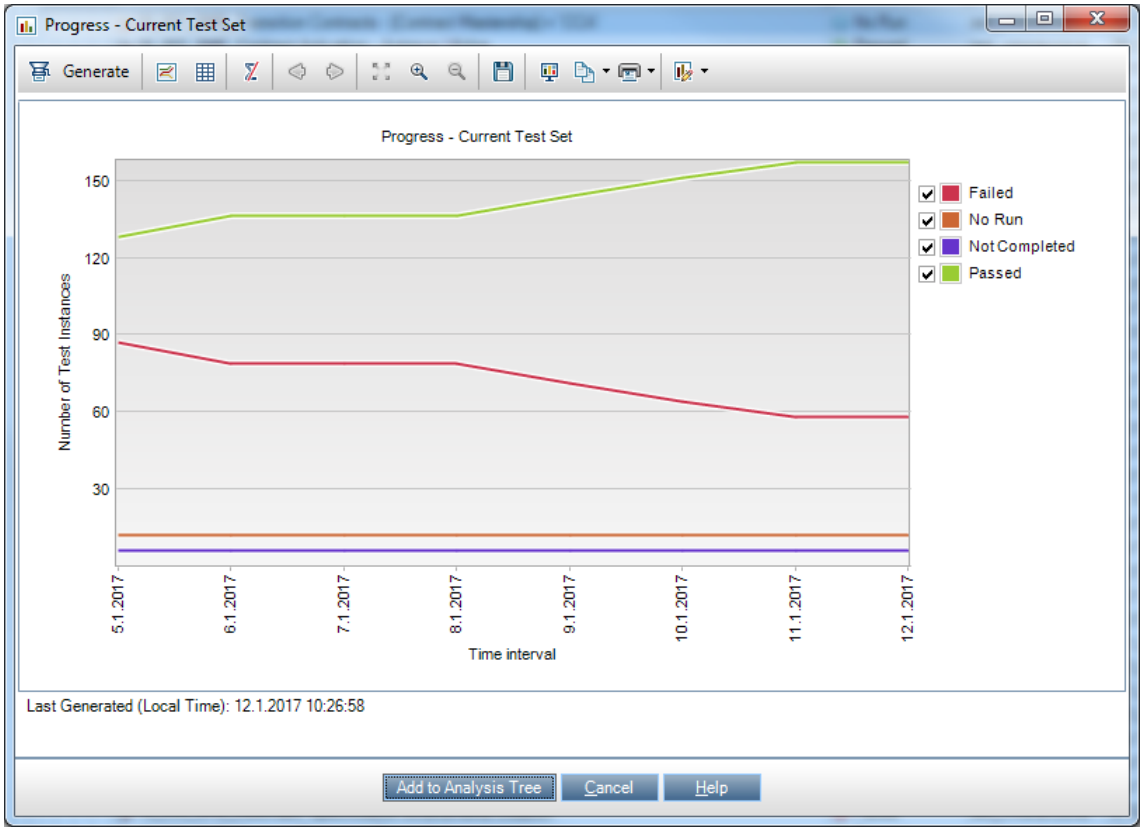
5.1 Znázornění regresních testů v průběhu testovacích kol

Tato podkapitola se zabývá hlavním cílem bakalářské práce a ukazuje míru užitečnosti testování softwaru v praxi na problematice regresního testování softwaru.

5.1.1 Sběr dat při regresním testování

Naším zadáním je otestovat aplikaci internetového bankovnictví ve 4 kolech regresního testování, jejich výsledky promítnout v grafu a dále je zpracovat pomocí základních statistických metod a procedur.

Na následujícím obrázku (Obrázek 7) je zobrazen pokrok testovacích scénářů v průběhu osmi po sobě následujících dnů, pomocí progresivního grafu, který je výstupem z nástroje HP Quality center. Zelená křivka znázorňuje průběh testovacích scénářů, které skončily úspěšně a nebyla v nich nalezena žádná chyba. Naopak červená křivka zobrazuje průběh testovacích scénářů, které skončily s alespoň jednou chybou.



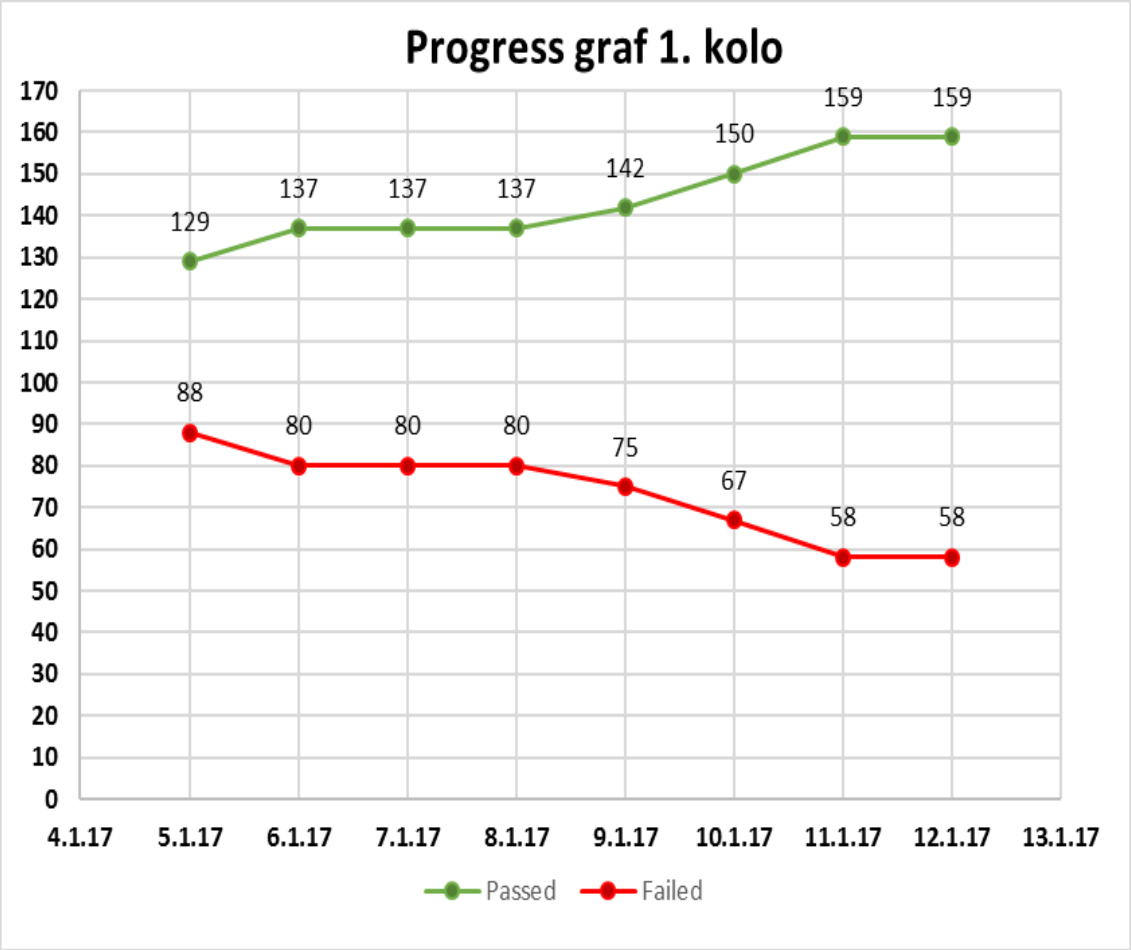
Obrázek 7: První kolo – progres graf v nástroji HP Qc. Zdroj: (vlastní zpracování).

1. kolo		
Datum	Passed	Failed
5.1.2017	129	88
6.1.2017	137	80
7.1.2017	137	80
8.1.2017	137	80
9.1.2017	142	75
10.1.2017	150	67
11.1.2017	159	58
12.1.2017	159	58

Tabulka 2: Data prvního kola testování. Zdroj: (vlastní zpracování).

Získaná data při regresním testování daného kola jsou zapsána do tabulek, které znázorňují úspěšné a neúspěšné testy v jednotlivých dnech.

Pro přehlednější zobrazení jsou data zpracována v programu Excel a jejich průběh zobrazen na bodovém grafu s rovnými spojnicemi a značkami (Obrázek 8, Obrázek 9, Obrázek 14, Obrázek 15).



Obrázek 8: První kolo – progres graf. Zdroj: (vlastní zpracování).

Na grafu (Obrázek 8) je vidět, že křivka úspěšných testovacích scénářů, které skončily bez softwarové chyby, má rostoucí tendenci a v průběhu týdne se z chybových scénářů stávají úspěšné. Dále je vidět, že graf je souměrný podle osy x, která zobrazuje průběh testů v čase, tedy v jednotlivých dnech. Souměrnost je zapříčiněna díky 100% otestovanosti testovacích scénářů.

Chybových testovacích scénářů, které skončily s alespoň jednou chybou, ubývá a chyby jsou v průběhu jednotlivých dnů opravovány. Poté se posílají zpět softwarovým testerům na takzvaný retest, kde se ověří, zdali byla chyba opravena a pokud ano, scénář je označen jako úspěšný.

K přechodu na další kolo dochází za předpokladu, že všechny testy byly dokončeny s nějakým konkrétním výsledkem. Pro přehlednost tohoto grafu počítáme

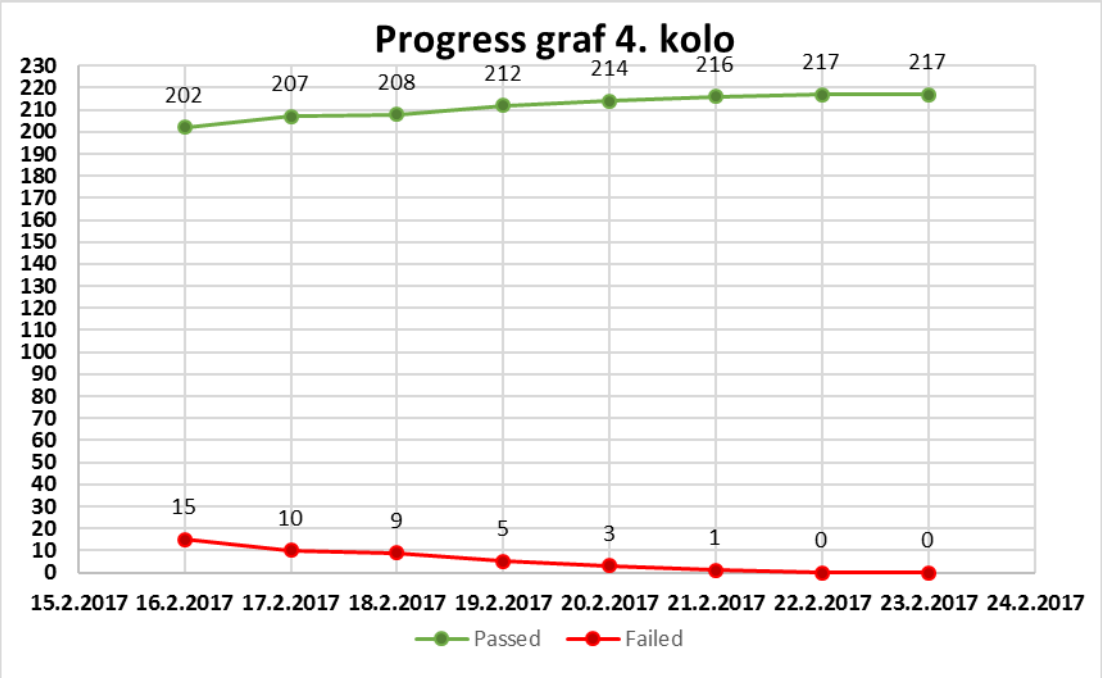
buď s výsledkem Passed – testovací scénář skončil bez chyby nebo Failed – testovací scénář skončil s alespoň jednou chybou.

Mezi jednotlivými koly je vždy prostor, v našem případě 6 dní, který je věnován kompletnímu otestování všech scénářů v dané sadě testů. V našem případě sada obsahuje 217 scénářů.

Průběh následujících kol má podobný průběh jako v prvním kole. To znamená, že počet chybových scénářů má klesající tendenci a počet úspěšných scénářů rostoucí tendenci, ale liší se svými hodnotami. Výsledky 2. a 3. kola jsou uvedeny v kapitole 9 Přílohy – 9.1 Obrázky a tabulky.

4.kolo		
Datum	Passed	Failed
16.2.2017	202	15
17.2.2017	207	10
18.2.2017	208	9
19.2.2017	212	5
20.2.2017	214	3
21.2.2017	216	1
22.2.2017	217	0
23.2.2017	217	0

Tabulka 3: Data čtvrtého kola testování. Zdroj: (vlastní zpracování).



Obrázek 9: Čtvrté kolo – progres graf. Zdroj: (vlastní zpracování).

Výsledky 4. (posledního) kola testování jsou znázorněny v předchozí tabulce (Tabulka 3) a v grafu (Obrázek 9).

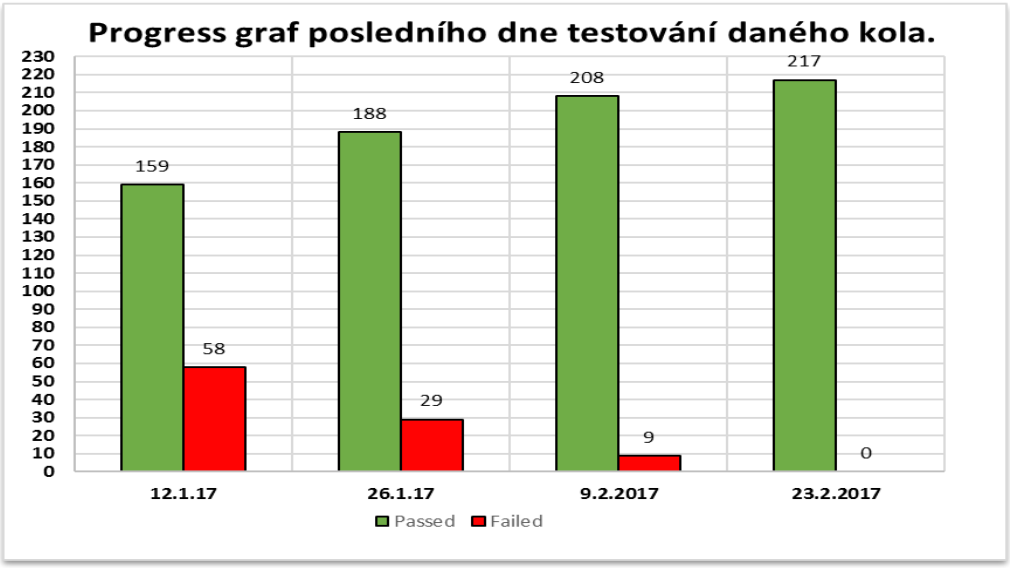
Z těchto dat lze vyčíst, že stav chybových scénářů (Failed) k 22. 2. 2017 je na čísle 0 a poslední den 23. 2. 2017 je tedy pouze formalita. V tento den stav chybových scénářů zůstává stále na čísle 0.

Toto ovšem neznamená, že testovaný program je bez chyby. Z grafů předchozích kol (Obrázek 8, Obrázek 14, Obrázek 15) lze vyčíst nárůst chybových scénářů v prvním dni testování oproti poslednímu dni testování předchozího kola. Aby bylo potvrzeno, že počet chybových scénářů je opravdu 0, muselo by být provedeno ještě alespoň jedno kolo regresního testování.

Pro názornost je uvedena tabulka (Tabulka 4) společně s grafem (Obrázek 10), napříč jednotlivými koly, které znázorňují výsledky na konci daného testovacího kola. Tato data jsou promítnuta do skupinového sloupcového grafu – Obrázek 10.

Výsledky všech kol		
	Počet passed	Počet failed
1. kolo	159	58
2. kolo	188	29
3. kolo	208	9
4. kolo	217	0

Tabulka 4: Výsledky posledního dne testování daného kola. Zdroj: (vlastní zpracování).



Obrázek 10: Graf výsledků všech kol. Zdroj: (vlastní zpracování).

Z předchozího grafu je vidět, že testovacích scénářů, které skončily s chybou, v průběhu času opět ubývá a v poslední testovací den (23. 2. 2017) je jejich stav na čísle 0. Toto je ideální stav regresního testování, kterého by na jeho konci mělo být dosaženo, tedy 100% testů ve stavu Passed a 0% testů ve stavu Failed.

V praxi bohužel dochází ke stavu, kdy nejsou všechny softwarové chyby opraveny a kvůli tomu je buď posunut termín dokončení testování projektu, nebo je produkt nasazen s chybou, na kterou je upozorněno a k její opravě dojde například pomocí opravného balíčku nebo při nasazení další verze. Neměl by však být vypuštěn do produkce produkt, který obsahuje závažnou softwarovou chybu, která například blokuje některou část programu.

5.1.2 Výsledky statistických procedur

Získaná data mohou být z testovacích nástrojů snadno exportována do Statistických nástrojů. V našem případě je využit nástroj SAS 9.4.

Mezi základní procedury se řadí procedura Means (Tabulka 5), která nám poskytne základní informace o zkoumaném souboru. Vstupem této procedury je počet neúspěšných testů po dobu 8 dnů pro jednotlivá kola. Hodnoty neúspěšných testů (Failed) v jednotlivých kolech jsou značeny (f1-f4). Výsledkem je pak průměr, směrodatná odchylka, minimální a maximální počet chybových scénářů na konci kola.

Procedura MEANS						
Analysis Variable : pocet						
failed	**N Obs**	**N**	**Mean**	**Std Dev**	**Minimum**	**Maximum**
f1	8	8	73.25	11.09	58.00	88.00
f2	8	8	50.75	16.02	29.00	74.00
f3	8	8	19.25	7.70	9.00	31.00
f4	8	8	5.38	5.48	0.00	15.00

Tabulka 5: Procedura Means. Zdroj: (vlastní zpracování).

Průměr v jednotlivých kolech má klesající tendenci, ale jinak nám tato charakteristika moc neřekne.

Daleko důležitější jsou pro nás hodnoty ve sloupci Std Dev, tedy hodnoty směrodatné odchylky. Tento údaj nám ukazuje, jak moc jsou hodnoty chybových scénářů rozptýleny nebo odchýleny od průměru hodnot. Nejvyšší hodnota směrodatné odchylky je ve 2. kole testování (řádek f2), s hodnotou 16,02 a můžeme tak usoudit, že v tomto kole došlo k největšímu pokroku. Naopak nejmenší pokrok je v posledním 4. kole, s hodnotou 5,48.

Pomocí hodnot minima a maxima lze získat přehled o rozdílech v posledním dni regresního testování daného kola (tedy hodnotami minima) a prvním dni regresního testování (tedy hodnotami maxima).

Pro další pozorování je zvolena procedura Univariate (Tabulka 6), která ukazuje základní charakteristiky souboru. Soubor nyní obsahuje všechny úspěšné testy pro jednotlivé dny regresního testování. Vstup do programu SAS je uveden v příloze (9.1 Vstupy programu SAS 9.4.

Procedura UNIVARIATE			
Variable: pocet			
Moments			
N	32	**Sum Weights**	32
Mean	179.84375	**Sum Observations**	5755
Std Deviation	28.8657732	**Variance**	833.232863
Skewness	-0.3072479	**Kurtosis**	-1.4032317
Uncorrected SS	1060831	**Corrected SS**	25830.2188
Coeff Variation	16.0504734	**Std Error Mean**	5.10279599

Tabulka 6: Procedura Univariate. Zdroj: (vlastní zpracování).

Počet všech testovacích dnů je 32 a těmto počtům nejsou manuálně přiřazeny žádné váhy, tudíž každý den má automaticky hodnotu 1. Suma vah (v tabulce Sum Weights) je tedy také 32.

Hodnota šikmosti (Skewness) je pro naše data -0,3072479. Tato hodnota ukazuje na asymetrické rozložení a zápornou/levostrannou šikmost, což znamená, že většina získaných hodnot se nachází nad průměrem.

Hodnota špičatosti (Kurtosis) je pro naše data -1,4032317. Tato hodnota zobrazuje koncentraci hodnot v místě, a jelikož je se záporným koeficientem znamená to, že normální rozdělení je plošší.

Základní statistické míry			
Location		**Variability**	
Mean	179.8438	**Std Deviation**	28.86577
Median	187.0000	**Variance**	833.23286
Mode	137.0000	**Range**	88.00000
		Interquartile Range	52.50000

Tabulka 7: Základní statistické míry. Zdroj: (vlastní zpracování).

Ze základních charakteristik (Tabulka 7) je opět možné vyčíst hodnotu průměru (Mean) $\bar{x}$=179,8438. Prostřední hodnotu v souboru (medián) nám ukazuje řádek Median $\tilde{x}$=187 a řádek Mode (modus) nám ukazuje nejčastější hodnotu souboru, což je hodnota 137 a tato hodnota je naměřena 6., 7. a 8. ledna v prvním kole regresního testování (tabulka 2).

Hodnoty rozptylu (Variance) a směrodatné odchylky (Std Deviation) jsou stejné jako v předchozí tabulce (Tabulka 6).

Mezi další statistické míry patří variační rozpětí (Range) s hodnotou 88. Ukazuje nám rozdíl mezi maximální a minimální hodnotou. Z této hodnoty lze určit počet chybových scénářů na počátku testování. Tento stav lze zjistit pouze za předpokladu, že stav chybových scénářů na konci testování je na čísle 0, což je v našem případě splněno.

Poslední statistickou mírou je interkvartilové rozpětí (Interquartile Range), což je rozdíl mezi třetím a prvním kvadrantem. Jeho hodnota je 52,5.

Kvantily (Definice 5)	
Level	**Quantile**
100% Max	217.0
99%	217.0
95%	217.0

Kvantily (Definice 5)	
Level	**Quantile**
90%	214.0
75% Q3	206.5
50% Median	187.0
25% Q1	154.0
10%	137.0
5%	137.0
1%	129.0
0% Min	129.0

Tabulka 8: Kvantily. Zdroj: (vlastní zrpacování).

Hodnota min (Tabulka 8) nám ukazuje jaká je nejnižší naměřená hodnota úspěšných testů. Tato hodnota byla naměřena v první den regresního testování.

Naopak honota max (Tabulka 8) nám ukazuje jaká je nejvyšší naměřená hodnota úspěšných testů a tato hodnota byla naměřena v poslední a předposlední den regresního testování.

Dále z tabulky 8 lze vyčíst hodnoty kvartilů. První (dolní kvartil) $\tilde{x}_{0,25}$=154 a třetí (horní kvartil) $\tilde{x}_{0,75}$=206,5. Mezi nimi je uvedena prostřední hodnota souboru neboli medián $\tilde{x}$=187.

Ostatní hodnoty udávají jednotlivé kvantily.

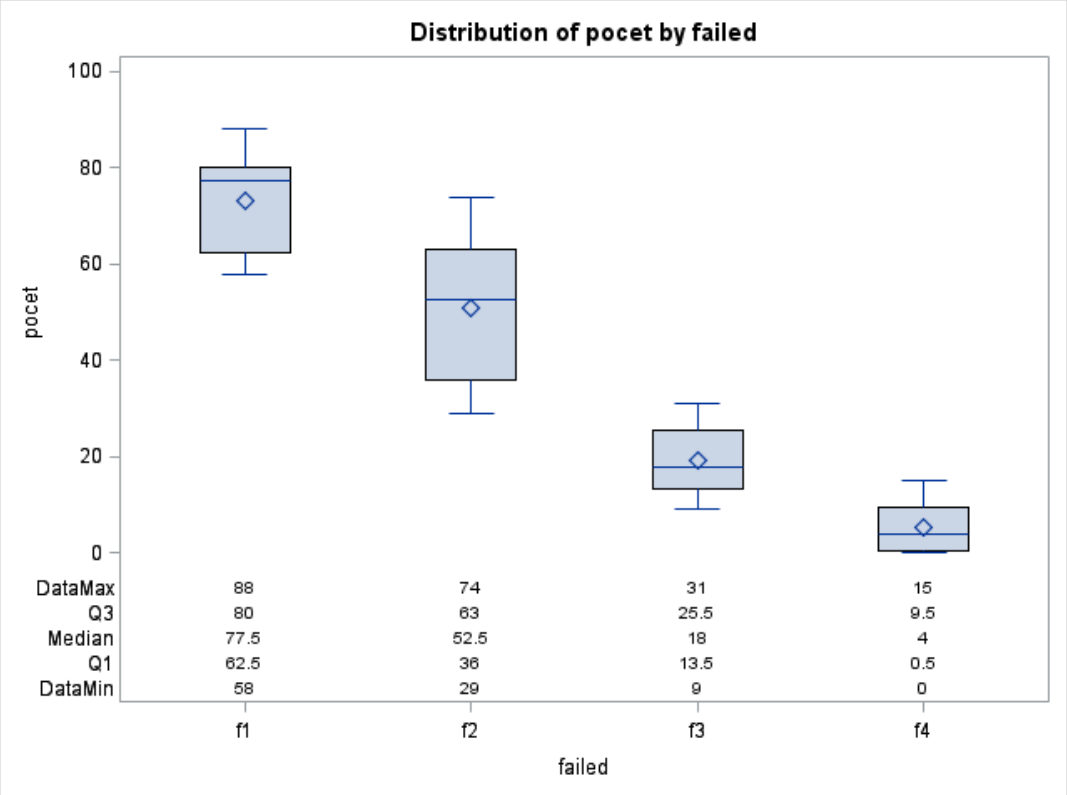
Na následující tabulce (Tabulka 9) je uvedeno ve sloupci (Lowest) pět nejnižších hodnot a ve sloupci (Highest) pět nejvyšších hodnot pozorování. Z této tabulky můžeme usoudit, že soubor není náchylný na extrémy a není tedy potřeba provádět úpravu souboru pomocí Winsorizace nebo pomocí Trimování.

Extrémní pozorování			
Lowest		**Highest**	
Value	**Obs**	**Value**	**Obs**
129	1	212	28
137	4	214	29

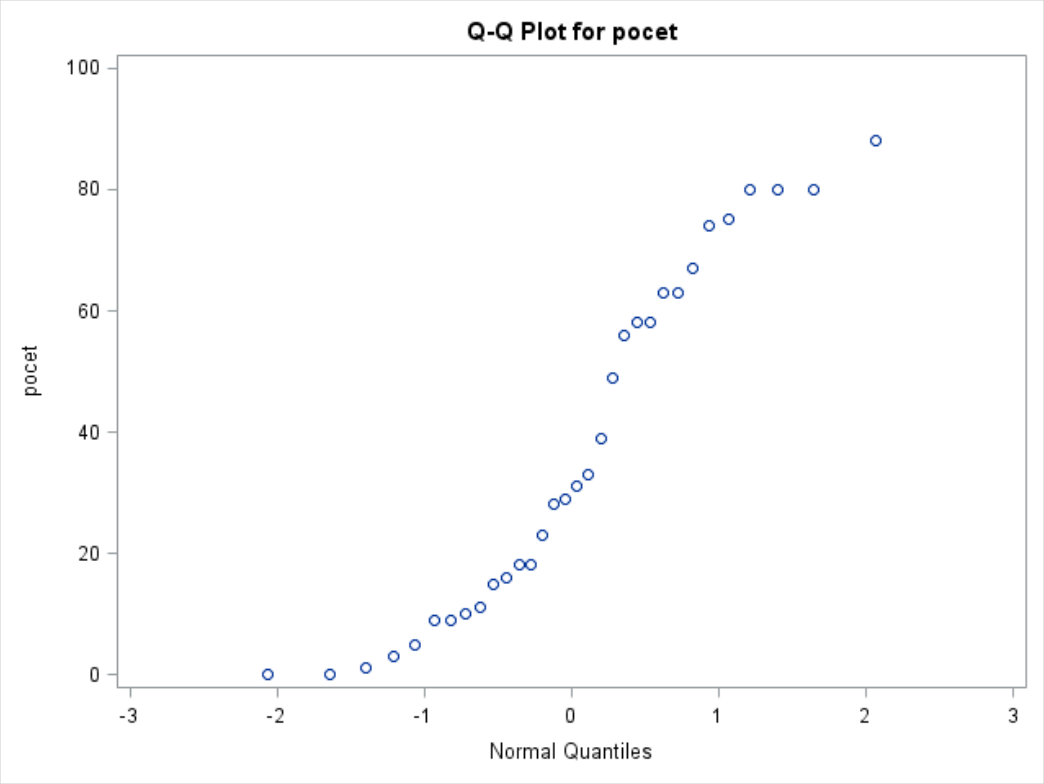
Extrémní pozorování			
Lowest		**Highest**	
Value	**Obs**	**Value**	**Obs**
137	3	216	30
137	2	217	31
142	5	217	32

Tabulka 9: Extrémní pozorování. Zdroj: (vlastní zpracování).

Procedurou Boxplot (Obrázek 11) vyvoláme graf, na kterém je přehledně zobrazen počet chybových scénářů, společně s hodnotami: minimum, dolní kvartil, median, horní kvartil, maximum.



Obrázek 11: Box plot - Krabicový graf. Zdroj: (vlastní zpracování).



Obrázek 12: Q-Q graf. Zdroj: (vlastní zpracování).

Z tohoto grafu (Obrázek 12) je patrné, že jednotlivé body jsou poměrně blízko pomyslné referenční přímky. Můžeme tedy konstatovat, že hodnoty mají normální rozdělení.

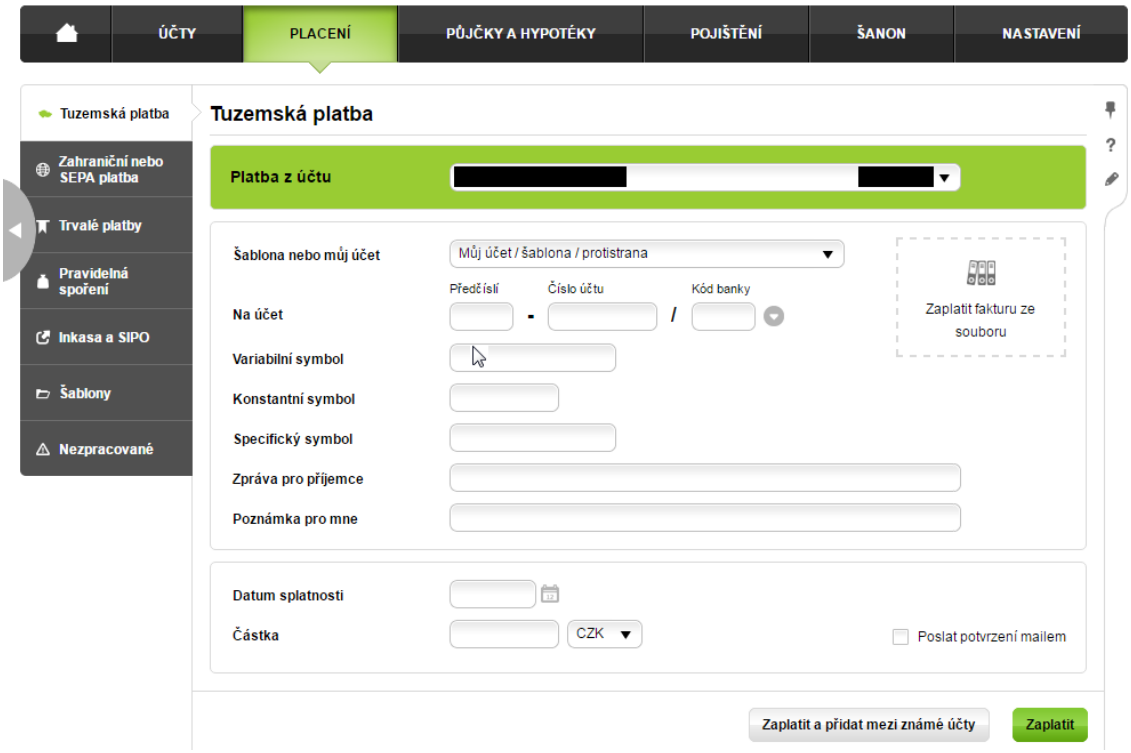
5.2 Vliv softwarové chyby na testy v průběhu testovacích kol

Tato kapitola názorně ukazuje regresní testování podrobněji, než tomu bylo v předchozí kapitole. Tabulky 10, 14, 15, 16 zobrazují průběh testovacích scénářů a jeho kroky při vzniku softwarové chyby v jednotlivých kolech. Příklad je použit z praxe, při testování internetového bankovnictví, konkrétně modulu tuzemské platby, vyplnění pole variabilní symbol.

ID případu užití	ID scénáře	Popis	Status	Defekt
UC0001	TS0001	Ověř funkci placení tak, že po stisku záložky PLACENÍ se zobrazí nabídky Tuzemská platba s předvyplněným polem Platba z účtu.	Passed	
UC0001	TS0002	Ověř textaci názvů polí (viz soubor s textacemi: textace_tuzemska platba.txt).	Passed	
UC0001	TS0003	Ověř, zdali se po najetí myší do pole Variabilní symbol zobrazí kurzor pro výběr textu (beam_r)	Failed	351
UC0001	TS0004	Ověř, zdali po kliku do pole Variabilní symbol se zobrazí blikající svislá čára.	Passed	
UC0001	TS0005	Ověř, zdali po kliku do pole Variabilní symbol se změní orámování pole na zelený rámeček.	Passed	
UC0001	TS0006	Přejdi do následujícího pole pomocí kliku na následující pole nebo stisknutím tlačítka TABulátoru a ověř, zdali se nezobrazila varovná hláška. (viz soubor s textacemi varovných hlášek: varovna hlaska_c3)	Passed	

Tabulka 10: Testovací sada modulu tuzemské platby – pole variabilní symbol. 1. kolo. Zdroj: (vlastní zpracování).

Softwarový tester prochází jednotlivé kroky testovacího scénáře a kontroluje jejich shodu s testovacím prostředím (Obrázek 13). Pokud v některém z kroků nalezne nesrovnalost testovacího prostředí s popisem, vytvoří softwarovou chybu (defekt).



Obrázek 13: Testovací prostředí Air Bank – pole variabilní symbol. 1. kolo. Zdroj: (www.ib.airbank.cz).

Průběh zbylých třech testovacích kol je uveden v příloze 9.2 Obrázky a tabulky - Vliv softwarové chyby na testy v průběhu testovacích kol.

Z výsledků testovacích scénářů je utvořena tabulka 11. Z této tabulky je vidět, že průběh v jednotlivých kolech se mění a především je zde vidět vliv softwarové chyby na ostatní kola regresních testů. Regresním testováním je zajištěna bezchybnost pole variabilní symbol v modulu tuzemské platby.

Pokud by byl využit klasický způsob testování softwaru, tedy testování v jednom kole, modul by nebyl bezchybně otestován. Chyba v 1. kole (Ověření, zdali se po najetí myší do pole Variabilní symbol zobrazí kurzor pro výběr textu) by sice byla odstraněna, ale nebyl by zjištěn její vliv na ostatní kroky daného testovacího scénáře nebo na jiné testovací scénáře.

Výsledky jednotlivých kol		
	Počet passed	Počet failed
1. kolo	5	1
2. kolo	4	2
3. kolo	5	1
4. kolo	6	0

Tabulka 11: Výsledky testovacích scénářů jednotlivých kol. Zdroj: (vlastní zpracování).

6 ZHODNOCENÍ VÝSLEDKŮ

Statistiky a výsledky jednotlivých testů a testovacích kol sleduje Test Analytik nebo Test Manažer a právě pro ně je důležité sledovat testovací statistiky a výstupy z testovacích nástrojů, na základě kterých se poté rozhoduje o budoucím plánování prací, lidských zdrojů a dalších opatřeních, například přesčasech.

Na základě výsledků jsou vypracována doporučení, která zlepšují a zefektivňují testovací práce. Tato doporučení jsou popsán v následujících bodech:

- **Vybrat vhodný testovací nástroj**

V podkapitole 4.9 srovnání testovacích nástrojů jsou uvedeny dva nejčastěji používané testovací nástroje.

Nejzásadnějším kritériem volby nástroje je velikost a složitost testované aplikace. Pokud jde o drobnou internetovou aplikaci nebo například web, k jeho otestování bohatě postačí nástroj Selenium.

Dalším kritériem je například cena, kterou jsme ochotni do testovacího nástroje investovat. Pokud je aplikace testována pro velkou společnost jako je například banka nebo telekomunikační společnost, je vhodné použít nástroj HP Quality center, který poskytuje spoustu nadstavbových funkcí a zajišťuje snadnou komunikaci napříč účastníky softwarového vývoje.

- **Sledovat vývoj testů**

Pro budoucí rozhodování na základě výsledků testů je důležité sledovat vývoj testů v čase. Tento vývoj lze sledovat v grafech, například v progres grafu nástroje HP Quality center, kde je názorně vidět průběh testů. Pokud mají chybové testy klesající tendenci, je to pro nás dobrá zpráva, že testovací kola probíhají v pořádku. Pokud nemají chybové testy klesající tendenci, je to varovný signál a je potřeba zjistit proč.

- **Sledovat softwarové chyby v testech a jejich priority**

Úkolem Test Analytik nebo Test Manažer, není jen sledování průběhu testů, ale také sledování průběhu softwarových chyb (defektů). K tomuto sledování také může pomoci testovací nástroj, který nám zobrazí jejich statistiky. V tabulkách (Tabulka 10, Tabulka 14, Tabulka 15, Tabulka 16) je ve sloupci defekt zobrazeno jejich identifikační číslo, pomocí kterého si k nim dohledáme všechny potřebné informace.

Ideální stav při zahájení dalšího kola je ten, že softwarová chyba je opravena a ve scénáři se již nevyskytuje. Proto je důležité sledovat jejich průběh, aby byly co nejdříve odstraněny a testování tak mohlo být ukončeno.

S každým založením softwarové chyby je vyplněna její priorita, na jejímž základě se upřednostňuje její odstranění.

Průběh chyb lze sledovat i na základě chybových testovacích scénářů. Musíme však počítat s tím, že u každého scénáře může být více chyb a tak po odstranění jedné chyby nemusí být scénáři změněn stav na úspěšný (Passed). Varovný signál může být i to, že se počet chybových scénářů zastaví na konkrétním počtu. V proceduře Univariate našich dat s chybovými scénáři (Tabulka 8), to lze zjistit například pomocí nejčastější hodnoty nazývané Modus (Mode). Z našeho příkladu například vidíme ustálení chybových scénářů na čísle 80 po dobu 3 dnů a můžeme tak dále jednat a zavádět opatření.

- **Počítat s nárůstem chyb na počátku testovacího kola**

Na počátku každého testovacího kola je vidět mírný nárůst chybových scénářů, oproti poslednímu dni testování předchozího kola (Obrázek 8, Obrázek 9, Obrázek 14, Obrázek 15). To je dáno zanesením nových chyb do programu při jejich opravě nebo jejich přehlédnutím v předchozích kolech. Pokud by měl softwarový tester při každé chybě kontrolovat všechny ostatní scénáře a hledat, zdali je opravená chyba neovlivnila, měl by několikanásobně více práce a projekt by se tak mnohonásobně prodražil. Takovýto postup je možné provádět pouze u opravdu malých projektů (aplikací), kde jsou testy snadno provedeny a kde se chyby vyskytují spíše zřídka. Bezchybná aplikace ale téměř neexistuje a vždy je možné něco najít, i když si myslíme, že je produkt dokonalý.

Testy jsou proto prováděny v několika kolech, kde s výsledkem nadcházejícího kola dochází k poklesu testovacích scénářů, které skončily s chybou. I toto je cílem regresního testování.

- **Využívat statistiky pro budoucí plánování regresního testování**

Ze statistik jednotlivých procedur lze vyčíst informace pro budoucí plánování regresního testování v jednotlivých kolech nebo pro plán testovacích prací pro další projekty.

Z našich dat pomocí procedury Means lze vyčíst, že Směrodatná odchylka byla největší ve 2. kole regresního testování. Nabývá hodnoty 16,02, což informuje o velké rozptýlenosti jednotlivých hodnot znaku kolem průměru. A lze tedy předpokládat podobný průběh u dalších projektů, připravit se na něj a zavést opatření.

Z výstupu extrémního pozorování (Extreme observations) procedury Univariate vidíme 5 nejnižších hodnot a 5 nejvyšších hodnot, v našem případě počtu chybových scénářů. Nejnižší hodnoty (0, 0, 1, 3, 5) se vyskytují na konci pozorování, tedy v posledních dnech (28.-31. den pozorování), nejvyšší hodnoty (75, 80, 80, 80, 88) jsou naměřeny v prvních pěti dnech pozorování. A lze tedy opět předpokládat podobný průběh u dalších projektů.

7 ZÁVĚR

Tato bakalářská práce shrnuje informace týkající se regresního testování a testování obecně a slouží jako podklad pro všechny účastníky softwarového vývoje.

Hlavním cílem práce bylo utvořit jakási doporučení pro Test manažery a Test Analytiky.

Ve vlastní práce byl sledován průběh 217 regresních testů ve 4 kolech a tyto testy byly dále statisticky pozorovány. Dále se zde zabývalo vlivem softwarové chyby na ostatní testy v průběhu testovacích kol. V této části byl použit testovací scénář zabývající se modulem tuzemské platby, vyplněním pole variabilní symbol. Tento scénář obsahoval 6 kroků a byl také testován ve 4 kolech regresního testování. Z obou těchto částí byly formulovány výsledky vlastní práce, na jejichž základě byl utvořen návrh na zlepšení a zefektivnění testovacích prací.

Z vlastní práce bylo například zjištěno, že pokud je nalezená softwarová chyba následně opravena, může ovlivnit další kroky testovacího scénáře nebo ovlivnit jiné testovací scénáře. Proto je nutné regresní testování rozdělit do několika testovacích kol, aby došlo k eliminaci softwarových chyb v programu.

Na základě utvořených doporučení je dále možné rozhodovat o budoucím plánování testovacích prací, lidských zdrojů a dalších opatřeních napomáhajících snadnému průběhu projektu a jeho zefektivnění.

8 SEZNAM POUŽITÝCH ZDROJŮ

8.1 Bibliografie

1. BURNS, David. *Selenium 2 Testing Tools: Beginner´s Guide.* Birmingham: Packt Publishing, 2012. ISBN 978-18-495-1830-7.
2. LEWIS, William E., VEERAPILLAI Gunasekaran. *Software testing and continuous quality improvement*. 2. vydání. Boca Raton: Auerbach Publications, 2005. ISBN 08-493-2524-2.
3. PAGE, Alan, JOHNSTON, Ken, ROLLISON, Bj. *Jak testuje software Microsoft*. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5.
4. PATTON, Ron. *Testování softwaru*. 2. vydání. Brno: Computer Press, 2002. ISBN 80-7226-636-5.
5. STEPHENS, Matt, ROSENBERG, Doug. *Testování softwaru řízené návrhem*. Brno: Computer Press, 2015. ISBN 978-80-251-3607-2.
6. BUREŠ, Miroslav, RENDA, Miroslav, DOLEŽEL, Michal, SVOBODA, Peter, GRÖSSL, Zdeněk, KOMÁREK, Martin, MACEK, Ondřej, MLYNÁŘ, Radoslav. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Praha: Grada Publishing, 2016. ISBN 978-80-247-5594-6.
7. OTTO, Jan. *Ottův slovník naučný: illustrovaná encyklopaedie obecných vědomostí*. Praha: J. Otto, 1904.
8. BLACK, Rex. *ISTQB Advanced Function Exam Preparation Guide.* USA: Rex Black Consulting Services, Inc., 2007. ISBN 977-81-873.
9. LAST, Mark, KANDEL, Abraham, BUNKE, Horst. *Artificial intelligence methods in software testing*. Singapore: World Scientific, 2004. ISBN 978-981-2794-758.
10. LEVY, Steven. *Encyclopedia Britannica: Graphical user interface (GUI),* 2014.
11. MALISHEVSKY, Alexey. *Computer Science Journal of Moldova: The general prioritization framework,* 2016. ISSN 1561-4042.
12. *829-1998: IEE Standard for Software Test Documentation.* New York: Institute for Electrical and Electronics Engineers, 1998. ISBN 978-07-381-1444-8.

13. *829-2008: IEEE Standard for Software and System Test Documentation*. New York: Institute for Electrical and Electronics Engineers, 2008. ISBN 978-07-381-5746-7.
14. DOUNGSA-ARD, Chartchai. *Ethos: Generation of software test data from the design specification using heuristic techniques: exploring the UML state machine diagrams and GA based heuristic techniques in the automated generation of software test data and test code*. Bradford: University of Bradford, 2011.
15. KÁBA, Bohumil, SVATOŠOVÁ, Libuše. *Statistické nástroje ekonomického výzkumu*. Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2012. ISBN 978-80-7380-359-9.

8.2 Webové zdroje

1. *Hyperthot*. Software Development Methodology [online]. Chapman, 1997 [cit. 2017-01-10]. Dostupné z: http://www.hyperthot.com/pm_sdm.htm.
2. *Software Testing Class*. [online]. 2016 [cit. 2017-01-10]. Dostupné z: http://www.softwaretestingclass.com/what-is-difference-between-test-cases-vs-test-scenarios.
3. *Geeks for Geeks*. [online]. 2013 [cit. 2016-10-31]. Dostupné z: http://www.geeksforgeeks.org/designing-use-cases-for-a-project/.
4. *Testování softwaru*. Portál zabývající se problematikou ověřování kvality software. Manuální i automatizované testování [online]. 2011 [cit. 2017-01-10]. Dostupné z: http://testovanisoftwaru.cz.
5. *Slovník cizích slov*. [online]. 2017 [cit. 2017-03-7]. Dostupné z: http://slovnik-cizich-slov.abz.cz.
6. *Infapartner*. Infa partner datová inegrace [online]. 2010 [cit. 2017-03-8]. Dostupné z: http://www.infapartner.cz.
7. *Tutorialspoint.* Symply easy learning [online]. 2017 [cit. 2017-03-8]. Dostupné z: https://www.tutorialspoint.com.

8.3 Seznam Obrázků

8.4 Seznam Tabulek

9 PŘÍLOHY

9.1 Vstupy programu SAS 9.4

```
data stat1;
        input passed $ pocet@@;
        datalines;
        p1 129 p1 137 p1 137 p1 137 p1 142 p1 150 p1 159 p1 159
        p2 143 p2 154 p2 154 p2 161 p2 168 p2 178 p2 184 p2 188
        p3 186 p3 189 p3 194 p3 199 p3 199 p3 201 p3 206 p3 208
        p4 202 p4 207 p4 208 p4 212 p4 214 p4 216 p4 217 p4 217
;
        proc means data=stat maxdec=2;
        class passed;
    var pocet;
        run;

        proc boxplot data=stat;
        plot pocet*passed/boxstyle=schematic;
        insetgroup min q1 q2 q3 max;
        plot pocet*passed/boxstyle=schematic
        notches;
        insetgroup min q2 max;
        run;

data stat2;
        input failed $ pocet@@;
        datalines;
        f1 88 f1 80 f1 80 f1 80 f1 75 f1 67 f1 58 f1 58
        f2 74 f2 63 f2 63 f2 56 f2 49 f2 39 f2 33 f2 29
        f3 31 f3 28 f3 23 f3 18 f3 18 f3 16 f3 11 f3 9
        f4 15 f4 10 f4 9 f4 5 f4 3 f4 1 f4 0 f4 0
;
        proc means data=stat maxdec=2;
        class failed;
    var pocet;
        run;

        proc boxplot data=stat;
        plot pocet*failed/boxstyle=schematic;
        insetgroup min q1 q2 q3 max;
        plot pocet*failed/boxstyle=schematic
        notches;
        insetgroup min q2 max;
        run;

        proc univariate data=stat1;
        histogram pocet/normal;
```

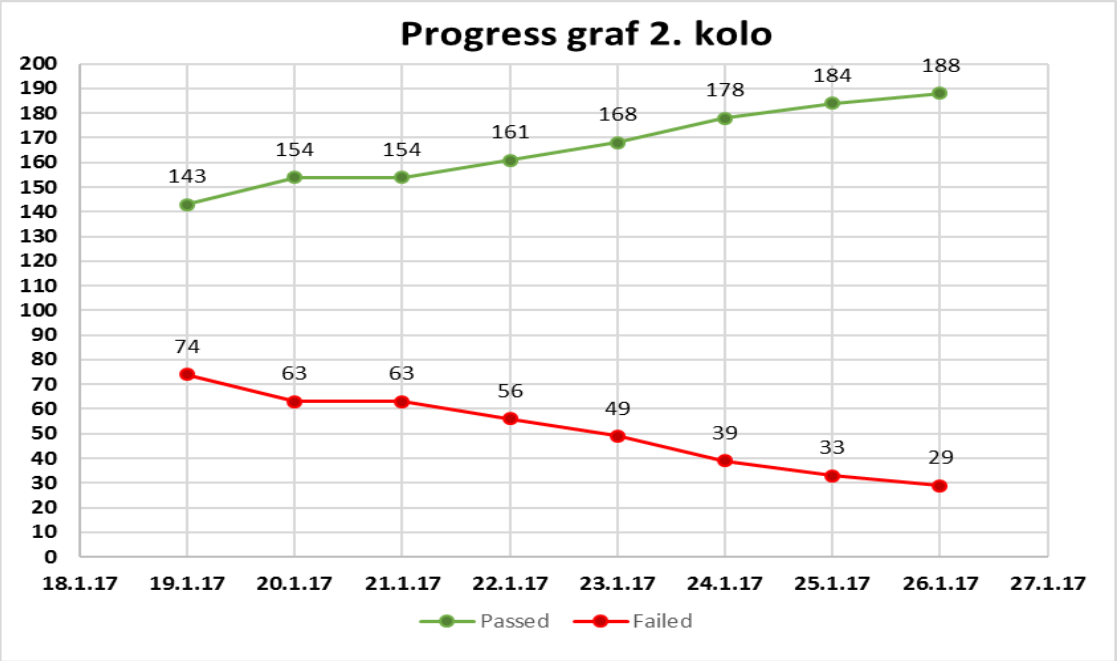
```
qqplot pocet/normal (mu=est);
var pocet;
run;

proc univariate data=stat2;
histogram pocet/normal;
qqplot pocet/normal (mu=est);
var pocet;
run;
```

9.2 Obrázky a tabulky

Znázornění regresních testů v průběhu testovacích kol:

2.kolo		
Datum	Passed	Failed
19.1.17	143	74
20.1.17	154	63
21.1.17	154	63
22.1.17	161	56
23.1.17	168	49
24.1.17	178	39
25.1.17	184	33
26.1.17	188	29

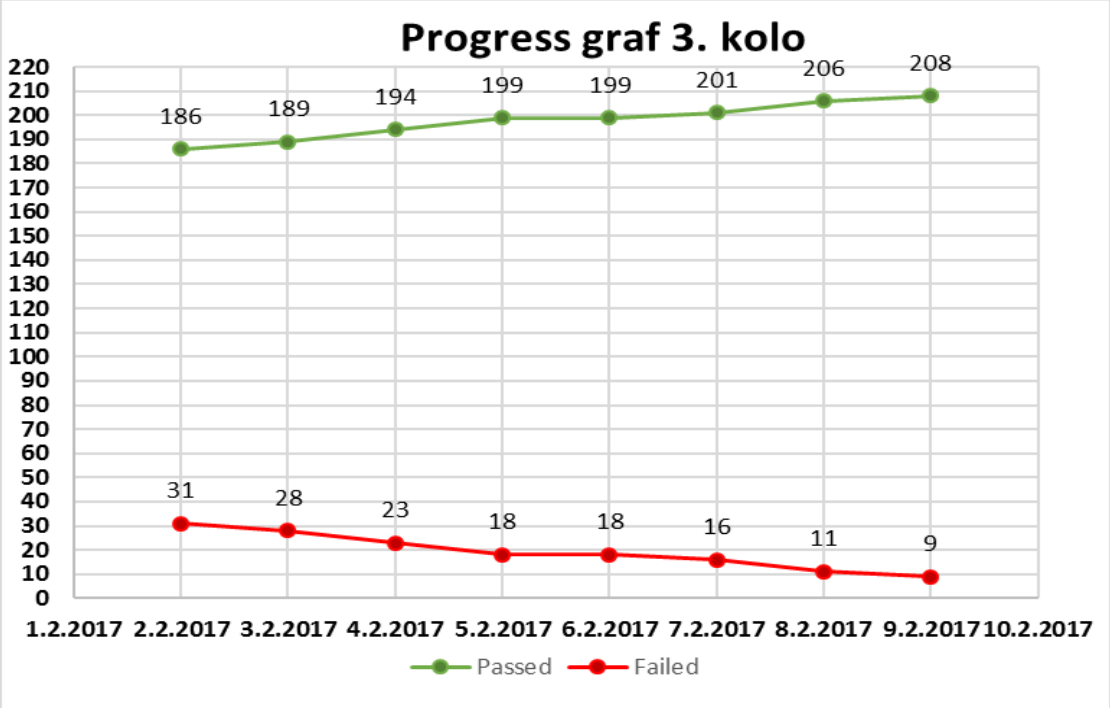
Tabulka 12: Data druhého kola testování. Zdroj: (vlastní zpracování).



Obrázek 14: Druhé kolo – progres graf. Zdroj: (vlastní zpracování).

3.kolo		
Datum	Passed	Failed
2.2.2017	186	31
3.2.2017	189	28
4.2.2017	194	23
5.2.2017	199	18
6.2.2017	199	18
7.2.2017	201	16
8.2.2017	206	11
9.2.2017	208	9

Tabulka 13: Data třetího kola testování. Zdroj: (vlastní zpracování).



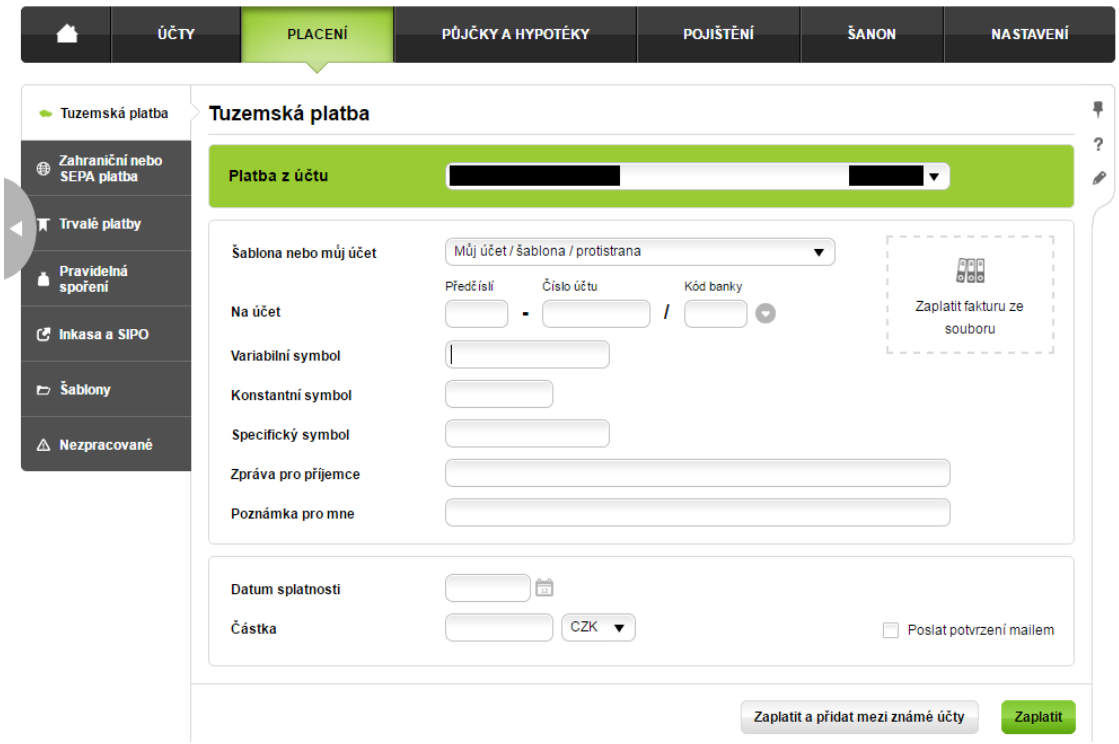
Obrázek 15: Třetí kolo – progres graf. Zdroj: (vlastní zpracování).

Vliv softwarové chyby na testy v průběhu testovacích kol:

ID případu užití	ID scénáře	Popis	Status	Defekt
UC0001	TS0001	Ověř funkci placení tak, že po stisku záložky PLACENÍ se zobrazí nabídky Tuzemská platba s předvyplněným polem Platba z účtu.	Passed	
UC0001	TS0002	Ověř textaci názvů polí (viz soubor s textacemi: textace_tuzemska platba.txt).	Passed	
UC0001	TS0003	Ověř, zdali se po najetí myší do pole	Passed	

		Variabilní symbol zobrazí kurzor pro výběr textu (beam_r)		
UC0001	TS0004	Ověř, zdali po kliku do pole Variabilní symbol se zobrazí blikající svislá čára.	Failed	482
UC0001	TS0005	Ověř, zdali po kliku do pole Variabilní symbol se změní orámování pole na zelený rámeček.	Failed	483
UC0001	TS0006	Přejdi do následujícího pole pomocí kliku na následující pole nebo stisknutím tlačítka TABulátoru a ověř, zdali se nezobrazila varovná hláška. (viz soubor s textacemi varovných hlášek: varovna hlaska_c3)	Passed	

Tabulka 14: Testovací sada modulu tuzemské platby – pole variabilní symbol. 2. kolo. Zdroj: (vlastní

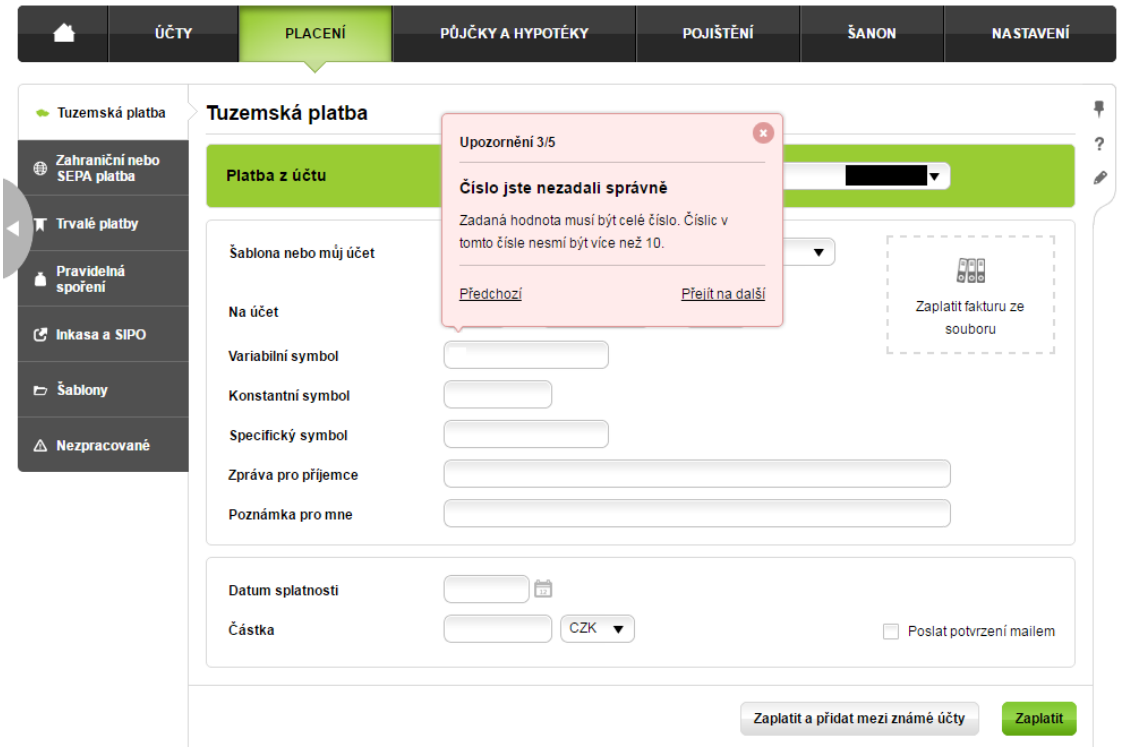


Obrázek 16: Testovací prostředí Air Bank – pole variabilní symbol. 2. kolo. Zdroj: (www.ib.airbank.cz).

ID případu užití	ID scénáře	Popis	Status	Defekt
UC0001	TS0001	Ověř funkci placení tak, že po stisku záložky PLACENÍ se zobrazí nabídky Tuzemská platba s předvyplněným polem Platba z účtu.	Passed	
UC0001	TS0002	Ověř textaci názvů polí (viz soubor	Passed	

		s textacemi: textace_tuzemska platba.txt).		
UC0001	TS0003	Ověř, zdali se po najetí myší do pole Variabilní symbol zobrazí kurzor pro výběr textu (beam_r)	Passed	
UC0001	TS0004	Ověř, zdali po kliku do pole Variabilní symbol se zobrazí blikající svislá čára.	Passed	
UC0001	TS0005	Ověř, zdali po kliku do pole Variabilní symbol se změní orámování pole na zelený rámeček.	Failed	497
UC0001	TS0006	Přejdi do následujícího pole pomocí kliku na následující pole nebo stisknutím tlačítka TABulátoru a ověř, zdali se nezobrazila varovná hláška. (viz soubor s textacemi varovných hlášek: varovna hlaska_c3)	Passed	

Tabulka 15: Testovací sada modulu tuzemské platby – pole variabilní symbol. 3. kolo. Zdroj: (vlastní zpracování).

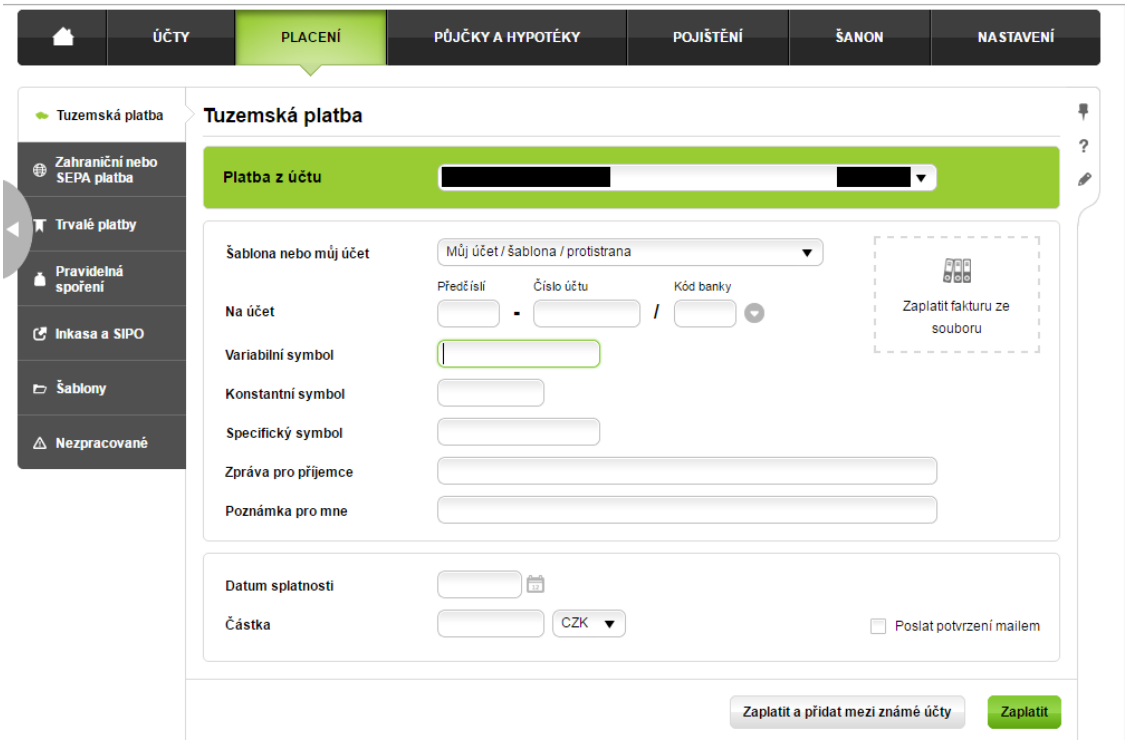


Obrázek 17: Testovací prostředí Air Bank – pole variabilní symbol. 3. kolo. Zdroj: (www.ib.airbank.cz).

ID případu užití	ID scénáře	Popis	Status	Defekt
UC0001	TS0001	Ověř funkci placení tak, že po stisku záložky PLACENÍ se zobrazí nabídky Tuzemská platba	Passed	

		s předvyplněným polem Platba z účtu.		
UC0001	TS0002	Ověř textaci názvů polí (viz soubor s textacemi: textace_tuzemska platba.txt).	Passed	
UC0001	TS0003	Ověř, zdali se po najetí myší do pole Variabilní symbol zobrazí kurzor pro výběr textu (beam_r)	Passed	
UC0001	TS0004	Ověř, zdali po kliku do pole Variabilní symbol se zobrazí blikající svislá čára.	Passed	
UC0001	TS0005	Ověř, zdali po kliku do pole Variabilní symbol se změní orámování pole na zelený rámeček.	Passed	
UC0001	TS0006	Přejdi do následujícího pole pomocí kliku na následující pole nebo stisknutím tlačítka TABulátoru a ověř, zdali se nezobrazila varovná hláška. (viz soubor s textacemi varovných hlášek: varovna hlaska_c3)	Passed	

Tabulka 16: Testovací sada modulu tuzemské platby – pole variabilní symbol. 4. kolo. Zdroj: (vlastní zpracování).



Obrázek 18: Testovací prostředí Air Bank – pole variabilní symbol. 4. kolo. Zdroj: (www.ib.airbank.cz).

9.3 Terminologický slovník

- **Případ užití**

Případ užití neboli use case je seznam akcí nebo kroků událostí, typicky definující interakci mezi rolí (účastníkem) a systémem k dosažení cíle. (Geeksforgeeks, 2013) Zastupuje určitou funkčnost systému.

Podle definice ISTQB je případ užití sled operací v dialogu mezi účastníkem a součástí nebo systémem s hmatatelným výsledkem, kde účastník může být uživatel nebo cokoliv, co si může vyměňovat informace se systémem. (Black, 2007, s. 79)

- **Testovací případ**

Testovací případ neboli test case popisuje konkrétní akce prováděné s určitou softwarovou komponentou a jejich očekávané výsledky. (Page, Johnston, Rollison, 2009, s. 209) Podle standardu IEEE 829:2008 by dokumentace testovacího případu měla obsahovat unikátní identifikátor, účel či zaměření testu, specifikaci vstupních podmínek, vlastností a dat, potřeby prostředí a závislosti na ostatních testovacích případech (IEEE Computer Society, 2008).

Testovací případ č.0000			
Název: Výstižný název testovacího případu			
Oblast		Podoblast	
Priorita	Typ	Frekvence	Doba trvání (minuty)
1	Funkční	Každé sestavení	2
Popis			
Účel testu: Počáteční podmínky: Kroky: 1. 2. 3 4. Očekávané výsledky: Poznámky			

Tabulka 17: Příklad šablony pro zadávání testovacích případů. Zdroj: (Page, Johnston, Rollison, 2009, s. 209).

-
-
-

- **Testovací sada**

Testovací sada neboli Test set je podle Page a spol. soubor souvisejících testovacích případů nebo bodů. Sada testů je často jednotkou testované funkcionality omezená na danou komponentu či vlastnost. (Page, Johnston, Rollison, 2009, s. 214)

- **Testovací scénář**

Testovací scénář neboli test scenario je dokument, kde jsou testovací případy uspořádány do skupin tak, aby tvořily logický celek. Většinou se jedná o jedinou větu, která říká, co přesně testovat.

ID případu užití	ID scénáře	Testovací scénář	Počet testovacích případů
UC0001	TS0001	Ověř funkci přihlášení tak, že vyzkoušíš, zdali se do portálu může přihlásit pouze registrovaný zákazník.	6
UC0001	TS0002	Ověř nabídku umístění funkčnosti webu.	8

Tabulka 18: Příklad testovacího scénáře. Zdroj: (Software Testing Class, 2016).

- **Grafické uživatelské rozhraní (GUI)**

Počítačový program, který umožňuje lidem komunikovat s počítačem za použití symbolů, vizuálního prvku a zařízení polohy. (Levy, 2014)

- **Testovací dokumentace**

Testovací dokumentací rozumíme veškeré dokumenty spojené s realizací softwaru vzniklé v rámci projektu, ať už se jedná o manažerské výkazy nebo o uživatelskou či technickou dokumentaci softwaru.

Podle glosáře testovací dokumentace obsahuje:

- Systémový testovací plán
- Systémová design specifikace
- Systémová specifikace testovacích případů
- Systémové testovací procedury specifikací
- Systémové testovací záznamy
- Systémové testovací zprávy incidentů
- Systémový testovací souhrn zpráv

- **Glosář**

Slovník s výkladem nejasných slov nebo jejich překlady. (Slovník cizích slov, 2017)

- **Maskování dat**

Vývoj nových verzí aplikací a jejich testování vyžaduje reálná data, na základě kterých lze důkladně ověřit funkčnost a odstranit případné chyby před uvedením do provozu. Většina společností pro tyto účely používá různé kopie provozních (tj. „ostrých") dat. Ještě v nedávné době bylo zcela běžné poskytovat vývojářům a testerům provozní data bez jakýchkoliv úprav. Dnes se již situace naštěstí mění, jelikož společnosti pociťují stale větší tlak na zabezpečení dat před únikem citlivých informací do nepovolaných rukou. (Infapartner, 2010)