Metody testování webových aplikací

Methods of web application testing

Bc. Radovan Hél

Diplomová práce
2014

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Radovan Hél**
Osobní číslo:	**A11534**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačové a komunikační systémy**
Forma studia:	**prezenční**
Téma práce:	**Metody testování webových aplikací**
Téma anglicky:	**Web Application Testing Methods**

Zásady pro vypracování:

1. **Představte metody testování softwarových aplikací, diskutujte jejich výhody a důvody použití.**
2. **Analyzujte tyto metody a navrhněte zefektivnění z pohledu webových aplikací.**
3. **Věnujte pozornost dokumentaci, automatickému testování a jeho významu.**
4. **Vyhledejte a popište nástroje využívané při softwarovém testování webových aplikací.**
5. **Proveďte průzkum u vybraných softwarových firem zda a jak testují svůj produkt.**
6. **Na základě výsledků průzkumu navrhněte vhodnou metodiku testování.**

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **Brian Hambling a Peter Morgan ... [et]. AL]. Software testing: an ISEB foundation. 2nd ed. London: British Computer Society, 2010. ISBN 978-190-6124-762.**
2. **WHITTAKER, James A a Peter Morgan ... [et]. AL]. How Google tests software: an ISEB foundation. 2nd ed. New Jersey: Addison-Wesley, c2012, xxvii, 281 s. ISBN 978-0-321-80302-3.**
3. **HLAVA, Tomáš. Testování softwaru [online]. 2012 [cit. 2014-01-27]. Dostupné z: http://testovanisoftwaru.cz/**
4. **SHINDE, Vijay. Software Testing Help [online]. 2007 [cit. 2014-01-27]. Dostupné z: http://www.softwaretestinghelp.com/**
5. **Software Testing Tutorial [online]. 2001 [cit. 2014-01-27]. Dostupné z: http://www.tutorialspoint.com/software_testing/**

Vedoucí diplomové práce:	**Ing. Radek Šilhavý, Ph.D.** Ústav počítačových a komunikačních systémů
Datum zadání diplomové práce:	**7. února 2014**
Termín odevzdání diplomové práce:	**27. května 2014**

ABSTRAKT

Tato práce se zabývá metodami testování webových aplikací, jejich výhody a nevýhody. Prezentovány jsou pravidla a principy pro testování softwarových produktů s ohledem na použití u webových aplikací, kde je kladen zvláštní důraz na zabezpečení. Představeny jsou také nástroje využívané v procesu testování, s názornou ukázkou jejich funkčnosti. Byly provedeny dvě případové studie na vybraných firmách, zabývající se vývojem softwarových aplikací. Proces testování v daných firmách je popsán a na jeho základě byla navrhnuta vlastní metoda testování s ohlédnutím na zásady popsané v teoretické části práce.

Klíčová slova: Metody testování webových aplikací, Zásady a principy testování, Typy testování, Testovací nástroje

ABSTRACT

This work discusses methods of web application testing, their advantages and disadvantages. The rules and principles for testing software products are presented with a respect to the use for web applications, where special emphasis on security is placed. Tools used in the testing process are also show nand thein functionality demonstrated. Two case studies on selected companies engaged in the development of software applications were conducted. The process of testing in these companies is described and the concept of it was used to design an own testing method with consideration on the principles described in the theoretical part.

Keywords: Methods of web application testing, Rules and principles of testing, Testing types, Testing tools

Na tomto místě bych rád poděkoval rodině za podporu při studiu a také mému vedoucímu diplomové práce Ing. Radku Šilhavému, Ph.D. za odborné vedení a cenné rady v průběhu vypracovávání této práce.

„A journey of a thousand miles begins with a single step.“

Lao-tzu, Čínský filozof (604 př.n.l. – 531 př.n.l)

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;

- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- Že odevzdaná verze diplomové/bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 26.5.2014

.....
podpis diplomanta

OBSAH

ÚVOD **10**
I TEORETICKÁ ČÁST **11**
1 ZÁKLADY TESTOVÁNÍ **12**
1.1 DEFINICE TESTOVÁNÍ 12
1.2 PROČ TESTOVAT 12
1.2.1 Softwarová chyba 13
1.3 KDY PŘESTAT TESTOVAT 14
1.3.1 Riziko 15
1.3.1.1 Projektové riziko 15
1.3.1.2 Produkční riziko 15
1.3.2 Verifikace a Validace 16
1.4 ZÁKLADNÍ TESTOVACÍ PROCES 16
1.5 TESTOVACÍ DOKUMENTACE 18
1.5.1 Testovací plán 18
1.5.2 Testovací scénář 18
1.5.3 Testovací případ 19
2 ŽIVOTNÍ CYKLUS VÝVOJE SOFTWARE **20**
2.1 MODELY 20
2.1.1 Vodopádový model 20
2.1.2 Iterační model 21
2.1.3 Programování řízené testy 23
2.1.4 V-model 24
3 PRINCIPY TESTOVÁNÍ **27**
3.1 TESTOVÁNÍ UKAZUJE PŘÍTOMNOST CHYB 27
3.2 ÚPLNÉ TESTOVÁNÍ JE NEMOŽNÉ 27
3.3 BRZKÉ TESTOVÁNÍ 28
3.4 SHLUKOVÁNÍ CHYB 29
3.5 PESTICIDE PARADOX 29
3.6 TESTOVÁNÍ JE KONTEXTOVĚ ZÁVISLÉ 30
3.7 KLAM ABSENCE CHYB 30
4 ÚROVNĚ TESTOVÁNÍ **31**
4.1 UNIT TEST: 31
4.2 INTEGRAČNÍ TESTOVÁNÍ 31
4.2.1 Integrace „Velký třesk“ 32
4.2.2 Integrace „Shora-dolů“ 32
4.2.3 Integrace „Zdola-nahoru“ 33
4.3 SYSTÉMOVÉ TESTOVÁNÍ 34
4.4 AKCEPTAČNÍ TESTOVÁNÍ 34
4.5 REGRESNÍ TESTOVÁNÍ 35
5 METODY A TYPY TESTOVÁNÍ **36**

5.1 METODY TESTOVÁNÍ 36
5.1.1 Black box 36
5.1.2 White box 37
5.2 TYPY TESTOVÁNÍ 37
5.2.1 Funkční testování 37
5.2.2 Nefunkční testování 38
5.2.2.1 Testování výkonu 38
5.2.2.2 Testování použitelnosti 39
5.2.2.3 Testování bezpečnosti 40
5.2.3 Manuální testování 41
5.2.3.1 Výhody manuálního testování 41
5.2.3.2 Nevýhody manuálního testování 41
5.2.4 Automatické testování 42
5.2.4.1 Výhody automatického testování 42
5.2.4.2 Nevýhody automatického testování 42
II PRAKTICKÁ ČÁST 44
6 TESTOVACÍ NÁSTROJE 45
6.1 JIRA 45
6.1.1 Základní prvky programu 45
6.2 SELENIUM IDE 47
6.2.1 Grafické rozhraní 48
6.2.2 Záznam a spuštění testovacího případu 49
6.3 OWASP ZED ATTACK PROXY 50
6.3.1 Obsluha programu 51
6.4 BROKEN LINK CHECKER 51
6.4.1 Výhody a nevýhody 52
6.4.2 Obsluha programu 52
7 PŘÍPADOVÉ STUDIE 55
7.1 FNZ S.R.O. 55
7.1.1 Popis testování 55
7.1.2 Metoda testování 56
7.1.3 Životní cyklus defektu v FNZ 58
7.2 AVG TECHNOLOGIES 59
7.2.1 Popis testování 59
7.2.1.1 Regular release 59
7.2.1.2 Operativní release 60
7.2.1.3 Emergency release 60
7.2.1.4 Postup testování 60
7.2.2 Rozdělení týmů 60
7.2.3 Schéma 62
7.2.4 Životní cyklus defektu v AVG 63
8 NÁVRH METODY TESTOVÁNÍ 65
8.1 POPIS TESTOVÁNÍ 65
8.2 SCHÉMA 68
ZÁVĚR 69

CONCLUSION **70**
SEZNAM POUŽITÉ LITERATURY **71**
SEZNAM OBRÁZKŮ **75**
SEZNAM TABULEK **76**

ÚVOD

V dnešní době uživatel očekává od webové aplikace tu samou funkčnost, bezpečnost, spolehlivost a flexibilitu jako od programu nainstalovaného v počítači. Internetové připojení je stále dostupnější a tak i nároky na kvalitu aplikace přístupné přes internetový prohlížeč rostou. Data se stále více přesouvají na cloudové služby, lidská práce i zábava stále více závisí na komunikaci přes e-maily, sociální sítě a různé komunikátory. Firmy hledají způsob jak jednoduše implementovat a rozmístit software v organizaci – ať už se jedná o účetnictví, management anebo služby zákazníkům. Tohle vše vytváří tlak na vývojáře k vytvoření webové aplikace, která bude nejen splňovat funkční nároky, bude intuitivní, spolehlivá a lehce dostupná, ale také poskytovat určitou úroveň uživatelského pohodlí pro práci i zábavu.

Proto je nesmírně důležité webovou aplikaci důkladně otestovat před jejím ostrým spuštěním. Vytvoření spolehlivé aplikace, se kterou se pracuje efektivně a působí příjemně na uživatele, buduje důvěru se zákazníkem. Spokojený zákazník se bude častěji k aplikaci vracet a doporučovat ji třetím stranám. Správně otestovaná aplikace také redukuje množství hlášených chyb od zákazníka a tím vede ke snížení nákladů a lidské práce při jejich odstraňování.

Webové aplikace se vyznačují zvláštnostmi, které je odlišují od jiných softwarových aplikací. Tyto zvláštnosti mohou ovlivnit jejich testování několika způsoby, které mohou mít za následek náročnější testování než v případě tradičních aplikací. Do značné míry pro ně však platí ty samé pravidla a principy a je tedy vhodné metody a techniky předem definovat a použít pro testování efektivně.

V této práci jsou popsány pravidla, principy a metody, jak správně a důkladně aplikaci otestovat. Správné provedené testování šetří nejen čas a náklady, ale taky umožňuje vytvoření kvalitnějšího výsledného produktu. V této práci jsou také představeny nástroje vhodné pro testování a popsány dva případy, jak testování probíhá v praxi. Na závěr je prezentován návrh procesu testování, který spojuje to nejlepší z obou případových studií s ohledem na pravidla a principy uvedené v teoretické části.

I. TEORETICKÁ ČÁST

1 ZÁKLADY TESTOVÁNÍ

1.1 Definice testování

Pod pojmem testování softwaru se často vybaví manuální procházení aplikace a klikání na nabídky za účelem odhalení defektu. Správné testování je však složitější a je pro něj zavedeno několik definic.

Předpoklad najít nedostatky v softwaru je správný, a na testování můžeme pohlížet jako na proces provádění programu nebo systému s úmyslem najít chyby. Toto testování se též označuje jako dynamické. Je totiž potřeba mít program spuštěný a mít k němu nějaké požadavky specifikující co má program dělat a jak se má chovat. [17]

O testování můžeme také říci, že zahrnuje jakoukoli činnost zaměřenou na hodnocení atributu či schopností programu nebo systému a určení, zda plní požadavky. Tato definice bere v potaz i statické testování, kdy není nutné mít program spuštěný. Často se jedná například o analýzu kódu. [9]

Testování je tedy proces ověřování kvality a jedna z nejpopulárnějších definic například uvádí, že testování softwaru je empirické šetření prováděné za účelem poskytnutí zúčastněným stranám informace o kvalitě testovaného produktu. [15]

Definice dle IEEE říká, že softwarové testování je proces analýzy softwaru k odhalení rozdílů mezi existujícími a požadovanými podmínkami, Bugů, a k zhodnocení jeho vlastností. [30]

1.2 Proč testovat

Žádný software není dokonalý, je totiž napsán lidmi a lidé dělají chyby. Následující příklady to jen potvrzují.

- První start rakety Ariane 5 Evropské Vesmírné Agentury v červnu 1996 selhal po 37mi sekundách. Softwarová chyba způsobila, že se raketa odklonila od jejího vertikálního vzestupu. Sebedestrukční schopnosti rakety byly aktivovány předtím, než nepředvídatelná dráha letu způsobila větší problémy.
- V listopadu 2005 byly zveřejněny informace o 10 nejhledanějších britských zločincích na webové stránce. Tato informace byla dopředu avizována v médiích a vyústila v nadměrný počet přístupů na stránku. Schopnosti stránky pojmout tolik přistupujících uživatelů byly nedostatečné a stránka byla stažena. [3]

- Na přelomu roku 1994 a 1995 byla vydána první multimediální hra pro děti - Disney´s The Lion King Animated Storybook. Hra byla silně inzerována a prodeje byly na tehdejší dobu obrovské. Nicméně den po Vánocích začaly zvonit telefony technické podpory a brzy byli technici zavaleni stížnosti rozzlobených rodičů o nemožnosti zprovoznění programu. Ukázalo se, že kompatibilita programu s nejrozšířenějšími PC modely na trhu byla nedostatečně otestována.

Softwarová chyba tedy může vést od jistého nepohodlí při používání přes nadbytečné finanční náklady až ke ztrátám na životech.

1.2.1 Softwarová chyba

V předchozích případech bylo selhání jasně patrné. Program nepracoval, jak bylo zamýšleno. Chyby však vždy nejsou tak jasně viditelné. Většinou jsou to jednoduché, nenápadné vlastnosti, někdy tak malé, že není jasné, zda se o chybu opravdu jedná.

Nazývat odchylkou od specifikace chybu, která zapříčiní pád rakety za několik milionů je stejně nepatřičné, jako označit překlep v textu jako selhání programu. Proto je zavedeno několik pojmů, které se vztahují k softwarové chybě. Jejich význam a využívání vždy závisí na konkrétním firemním prostředí a jejich zvyklostech. [22]

Přesto je však zavedeno několik obecných pojmů, které se více méně dodržují.

Error (Chyba): Má příčinu v lidské činnosti. Chyba může být v nestandardním kódu, jako chyba v syntaxi nebo ve volání proměnné. Může být také způsobená ve špatně nastaveném připojení k databázi.

Defect (Závada): Defekt je formální název bugu. Každý kdo se kdy zabýval testováním, se s tímto pojmem jistě setkal. K defektu dochází z důvodu nějaké chyby, například v kódu nebo v dokumentu, která způsobí odchylku od očekávaného výsledku. Defektem se tedy nazývá nalezená chyba. Pokud je Defekt potvrzen vývojářem nebo manažerem, stává se **Bugem**.

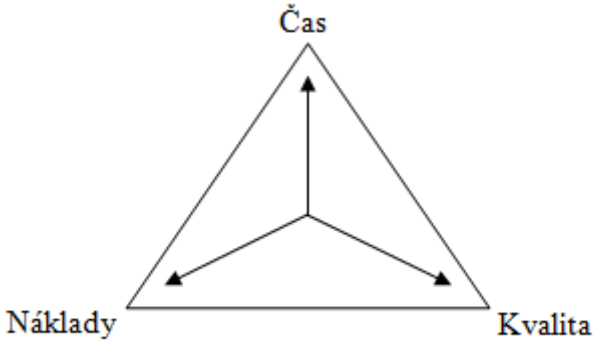
Failure (Selhání): Selhání je neschopnost systému nebo součásti vykonávat požadovanou funkci. Jinými slovy, je to závada zjištěná u koncového uživatele. Selhání může být způsobeno i vlivem prostředí, jako je radiace, magnetismus anebo obyčejná vlhkost. [16]

1.3 Kdy přestat testovat

Určit přesný moment kdy přestat testovat může být poměrně těžké. Většina novějších softwarových aplikací je tak složitá, a běžících v nezávislých prostředích, že úplné testování je nemožné. Běžně se tedy testování ukončuje na základě těchto faktorů:

- Testovací scénáře jsou dokončeny s určitou mírou úspěšnosti
- Pokrytí kódu/funkcionality/požadavků dosáhne specifického bodu
- Míra nacházení bugů klesne pod určitou úroveň
- Alfa nebo Beta testování končí
- Riziko a úroveň přijatelné kvality [10]

Jako každý jiný projekt i proces vývoje software musí dosáhnout kompromisu mezi kvalitou, času potřebnému na vývoj a náklady. V normální situaci je jeden z těchto faktorů stálý a zbylé dva se budou lišit v obráceném poměru k sobě navzájem. Nejčastěji bývá pevně dán čas, za který musí být aplikace dodána a kvalita se bude odvíjet od ceny dostupných zdrojů. Podobně může být definována úroveň kvality a cena vývoje se bude odvíjet od dostupného času. Pokud je času více, náklady bývají nižší a naopak. [13]



Obrázek 1 - Trojúhelník znázorňující vztah mezi projektovými náklady, kvalitou a časem [13]

1.3.1 Riziko

Rizikem rozumíme pravděpodobnost události, která vede k nežádoucím důsledkům ohrožujícím vývoj aplikace. Každý systém podléhá riziku a je zde jistá úroveň přijatelné kvality. Tyto faktory se využívají také při stanovování konce testování. Nejdůležitějším aspektem pro dosáhnutí přijatelné kvality aplikace v konečném a omezeném čase na testování je určení priorit. Stanovením priorit nám dá jistotu, že nejdůležitější a nejrizikovější prvky systému byly otestovány jako první.

1.3.1.1 Projektové riziko

Vyhodnocení projektového rizika se využívá pro správu schopnosti aplikaci dodat v požadovaném čase. Mezi tato rizika patří:

Problémy s dodávkou od třetích stran: Dodavatel může potřebný komponent či službu dodat pozdě, vůbec, anebo v nepřijatelné kvalitě.

Organizační činitele: Sem patří schopnosti testerů, školení a nedostatek personálu. Nedodržení návaznosti na informace nalezené při testování a hodnocení, neschopnost komunikace potřeb a výsledků.

Technické problémy: Potíže se sestavením správných požadavků. Včasné nepřipravení testovacího prostředí, špatná kvalita návrhu, kódu, konfigurace dat, testovacích dat, nebo testů.

Pro každé z rizik je určena pravděpodobnost jeho výskytu a následného dopadu. Stejně tak by měla být určena akce, která sníží šanci tohoto rizika vzniknout, anebo alespoň snížit jeho dopad, pokud se tak stane. Například když je určena pravděpodobnost, že dodavatel během projektu zkrachuje, jako střední (3 na stupnici od 1 do 5, kde 1 je riziko vysoké, 5 je riziko nízké) a dopad na projekt by byl velmi vysoký (1 na stejné stupnici) poté se úroveň rizika stanoví jako vysoká (1 * 3 = 3). Na základě této míry se poté manažer může rozhodnout, zda vůbec externí služby využít. [3]

1.3.1.2 Produkční riziko

Při plánování a vytváření testů může test leader použít přístup založený na produkčním riziku. Potenciální oblasti selhání softwaru jsou známy jako produkční rizika. Tato rizika ovlivňují kvalitu dodávaného produktu.

Produkční rizika: Dodání softwaru náchylného k chybám. Špatné softwarové vlastnosti (Nevyhovující funkcionalita, bezpečnost, spolehlivost, výkon). Aplikace neplnící své požadované funkce.

Hodnocení rizik se zde využívá k rozhodnutí, kdy a v jaké fázi životního cyklu vývoje software s testováním začít. Například riziko nedostatečné specifikace požadavků může být zmírněno použitím formálních kontrol hned, jak se v projektu začnou požadavky dokumentovat. Produkční rizika také poskytují informace umožňující rozhodnutí, kolik testování má být provedeno na konkrétní funkcionalitě nebo části systému. Čím větší riziko je stanoveno, tím detailnější a rozsáhlejší testování musí být. [3]

1.3.2 Verifikace a Validace

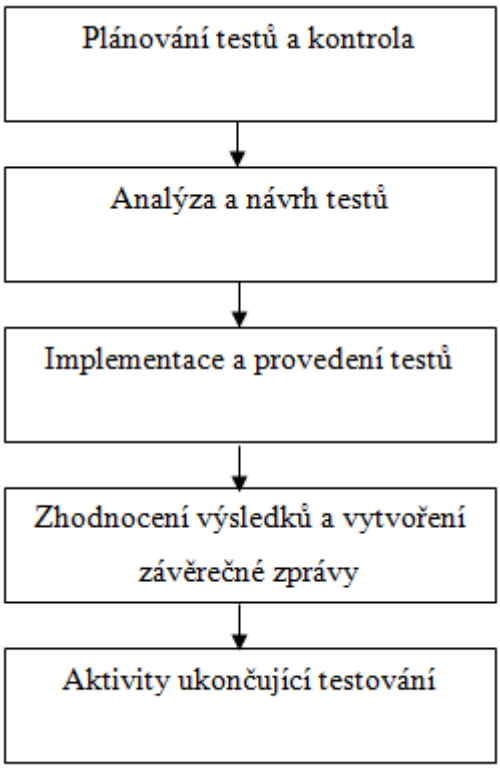
Verifikace a validace jsou nezávislé procedury zaměřené na prokázání, že systém odpovídá svým specifikacím a splňuje očekávání zákazníka.

Verifikace: Proces vyhodnocení, jestli výrobek, služba, nebo systém je v souladu s nařízeními, požadavky, specifikací nebo požadovaným stavem – kontrola, zda budujeme produkt správným způsobem. Často interní proces.

Validace: Proces ujištění, že výrobek, služba, nebo systém splňuje požadavky zákazníka či jiných zúčastněných stran – kontrola, zda budujeme ten správný produkt. Často vyžaduje přijetí externími zákazníky [23][27]

1.4 Základní testovací proces

Testovací proces nemá žádná pevná ohraničení nebo jasně danou posloupnost, přesto se hlavní aktivity vykonávají ve volně dodržovaném pořadí a jsou známy pod názvem Základní testovací proces. Je to klíčový prvek, co mají testeři dělat. Nejviditelnější část je spuštění testu. Proces však obsahuje i další aktivity jako plánování a příprava.



Obrázek 2 – Základní testovací proces [3]

Plánování testů a kontrola:

- Plánování testů je aktivita ověřující cíle testování, a specifikace testovacích činností, a testovacích kritérií za účelem dosažení stanovených cílů. Plánování bere v úvahu zpětnou vazbu od monitorovacích a kontrolních činností
- Kontrola je pokračující aktivita, která srovnává skutečný pokrok proti plánu a podává zprávy o stavu (včetně odchylek od plánu). To zahrnuje přijeté opatření nezbytných ke splnění úkolů a cílů projektu.

[5]

Analýza a návrh testů: Hlavními úkoly v této fázi je přezkoumání, zda máme k dispozici informace nutné k analýze a vytvoření tzv. *test case* (test case je soubor podmínek, na základě kterých tester určí, zda aplikace pracuje správně). Patří sem také zjištění testovacích podmínek, návrh testů, zhodnocení testovatelnosti jednotlivých požadavků a návrh testovacího prostředí s požadovanou strukturou a nástroji.

[11]

Implementace a provádění testů: Implementace a provádění testů zahrnuje aktivity spojené s kontrolou testovacího prostředí a spuštěním testů. Výsledky spuštěných testů musí být zaznamenány a veškeré rozdíly mezi očekávanými a skutečnými výsledky musí být vyšetřeny.

Zhodnocení výsledků a vytvoření závěrečné zprávy: Po provedení testů je provedena kontrola dosažených výsledků. Když byl například cíl stanoven, že z provedených testů bude 85% úspěšných, a dosaženo bylo pouze 75%, přichází na řadu rozhodnutí, zda se stanovené cíle změní, anebo se provedou další testy. Poté se vytvoří zpráva pro zúčastněné strany obsahující shrnutí toho, co bylo dosaženo, co bylo plánováno a obzvláště co testováno nebylo.

Aktivity ukončující testování: Po skončení samotného testování je třeba provést aktivity ujišťující, že všechny zprávy byly napsány, všechny defekty jsou uzavřené a ty, které jsou zatím odloženy do další fáze, musí být jasně označeny. Musí být provedena kontrola, že je veškerá dokumentace v pořádku. Dále se provádí archivace testovacího prostředí, infrastruktury a použitých testů. [3]

1.5 Testovací dokumentace

Testovací dokumentace zahrnuje dokumentaci artefaktů, které by měly být vypracovány předem, nebo v průběhu testovacího procesu.

Dokumentace v softwarovém testování usnadňuje odhad potřebného testovacího úsilí, pokrytí testy, sledování požadavků, atd. [33]

1.5.1 Testovací plán

Tento dokument popisuje rozsah, přístup, zdroje a plán zamýšleného testování. Uvádí mimo jiné testované položky, vlastnosti k otestování, kdo, co bude testovat, testovací prostředí a také kritéria pro začátek a ukončení testování. [3]

1.5.2 Testovací scénář

Testovací scénář nám říká, jaká oblast v aplikaci bude testována. Slouží k ověření, že žádná oblast nebude opomenuta a že všechny je celý proces otestován od začátku do konce. Každá testovaná oblast může mít jeden, nebo i několik stovek scénářů v závislosti na velikosti a složitosti aplikace.

1.5.3 Testovací případ

Testovací případy zahrnují sadu kroků, podmínek a vstupů, které mohou být použity při plnění úkolů testování. Hlavním cílem této aktivity je rozhodnout, zda aplikace splňuje požadavky na její funkčnost a další aspekty.

Dále se testovací případy používají pro sledování průběhu testování a jeho pokrytí. Není stanovena žádná formální šablona, nicméně užitečná dokumentace testovacího případu by měla obsahovat:

- ID testovacího případu.
- Modul aplikace.
- Verze aplikace.
- Účel.
- Předpoklady.
- Podmínky.
- Kroky.
- Očekávaný výsledek.
- Skutečný výsledek.

[33]

2 ŽIVOTNÍ CYKLUS VÝVOJE SOFTWARE

Metodologie vývoje software (také známá jako SDM) se neobjevila až do roku 1960. Životní cyklus vývoje systému (SDLC) může být považován za nejstarší formalizovaný rámec metodik pro budování informačních systémů. Hlavní myšlenkou SDLC je usilovat o vývoj informačních systémů velmi strukturovaným a metodickým způsobem. Vyžaduje od každé fáze životního cyklu, od vzniku myšlenky až po dodání finálního systému, aby byly provedeny důkladně a postupně ve zvoleném rámci. [4]

2.1 Modely

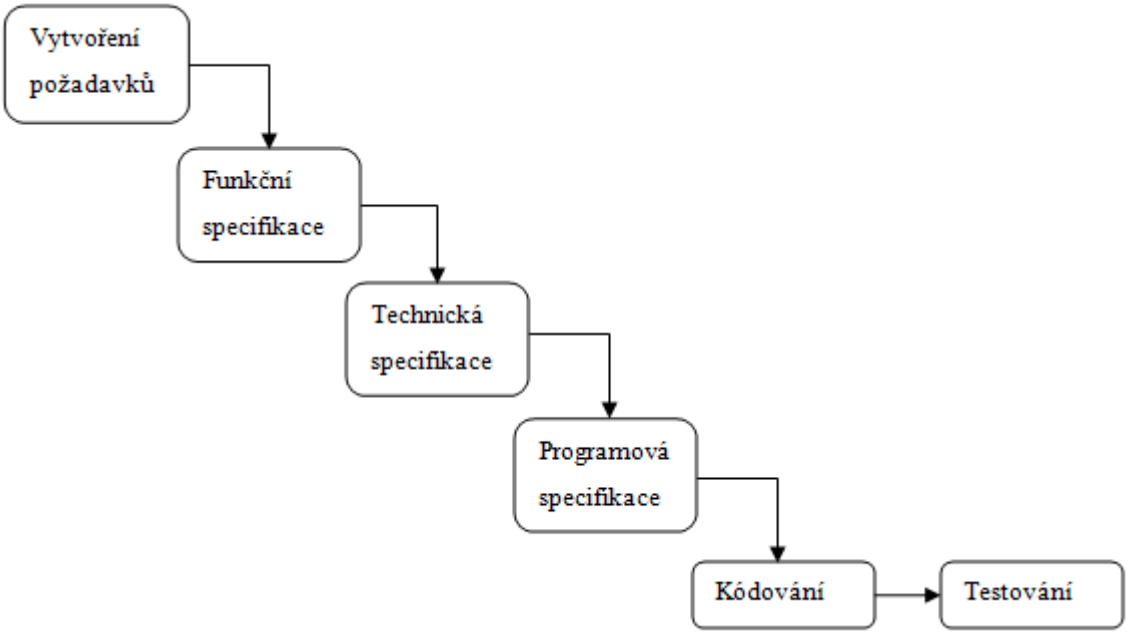
Model životního cyklu vývoje softwaru může být definován jako zjednodušený popis procesu, představený z určitého hlediska. V podstatě, je každá fáze softwarového procesu identifikovaná a model je poté navržen tak, aby představoval přirozené aktivity spojené s touto fází. V důsledku toho, sbírka „lokálních“ modelů může být využita k vytvoření obecného obrazu softwarového procesu. [26]

2.1.1 Vodopádový model

Vodopádový model je považován jako první model, který byl představen a široce využíván v softwarovém inženýrství. Jeho inovace spočívala v tom, že vývojový cyklus byl poprvé rozdělen do jednotlivých fází.

Dříve byly programy malé, s málo požadavky a jednoduše testovatelné. Jak se ale programy stávaly většími a komplexnějšími, vývojáři zjistili, že je čím dál komplikovanější si udržet představu o programu a tento obraz přenést do funkčního kódu. Také vznikla myšlenka na samostatnou tetovací fázi vykonávanou specializovanými testery. [29]

Vodopádový model má však velkou nevýhodu. Testování probíhá až na konci a slouží jako kontrola kvality. V tomto bodě může být produkt buď přijat anebo zamítnut. Ve vývoji software je ale nepravděpodobné, že můžeme jednoduše zamítnout části softwaru a vydat zbytek. Povaha funkčnosti softwaru je taková, že odříznutí části s chybami, může způsobit potíže v části jiné.



Obrázek 3 – Vodopádový model [3]

2.1.2 Iterační model

Tento model odvozuje svůj název od způsobu, jakým je software zkompletován. Při vývoji aplikace tímto způsobem je výsledný produkt složen jako řada dílčích sestavení skládajících se z vytvoření požadavků, návrhu, implementace a testování. V každé fázi iteračního vývoje je nová část kódována a poté začleněna do struktury, která je testována jako celek. Tímto způsobem se pokračuje až do splnění všech požadavků na danou aplikaci.

Tento model kombinuje prvky vodopádového modelu s iterativní filozofií. Iterační vývoj se zaměřuje na dodávky operačního produktu na konci každé iterace.

Příkladem tohoto přístupu je vývoj aplikace pro zpracovávání textu, kde jsou jednotlivé služby skládány v postupných buildech:

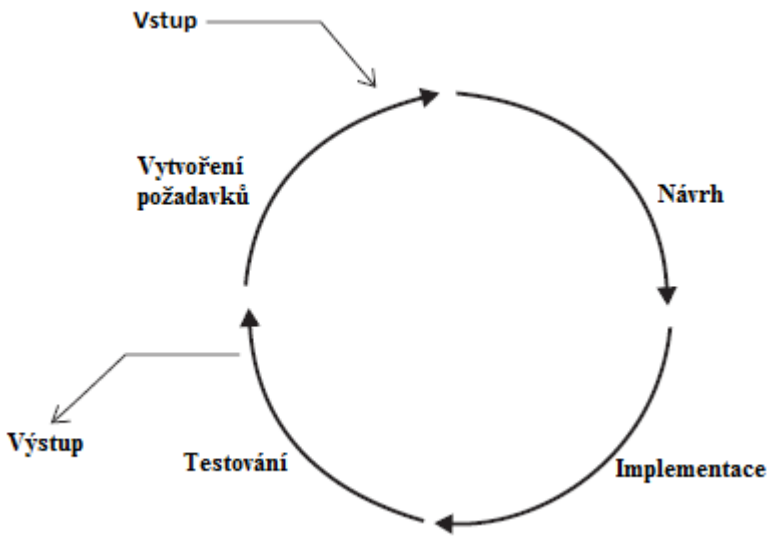
1. Základní správa souborů, editační a produkční funkce
2. Pokročilé úpravy textu
3. Kontrola pravopisu a gramatiky
4. Pokročilé rozložení stránky

Iterační vývoj je v nějaké své formě dnes nejběžnější postup pro vývoj aplikačních systémů. Tento vývoj může být buď řízen plánem, agilním přístupem, anebo, nejčastěji, jejich kombinací. Ve vývoji řízeném plánem jsou přírůstky systému určeny předem. V agilním přístupu jsou na počátku určeny přírůstky, ale vývoj v pozdějších krocích závisí na pokroku a prioritách zákazníka.

Z manažerského hlediska má však i tento vývojový model své nevýhody.

- Proces není vidět. Manažeři potřebují pravidelné výstupy k měření pokroku. Pokud jsou systémy vyvíjeny rychle, není nákladově efektivní produkovat dokumentaci, která by odrážela stav každé verze systému.
- Nedostatek formální dokumentace ztěžuje testování. Využitím vývoje řízeného testy může však být tento problém zmenšen.
- Struktura systému má tendenci degradovat s novými přidanými přírůstky. Pokud nejsou vynaloženy dodatečné prostředky na údržbu kódu, pravidelná změna vede poškození struktury, Začlenění dalších změn programu se stává stále více obtížné a nákladné.

Problémy iteračního vývoje se stávají zvláště akutní pro velké a komplexní systémy s dlouhodobou životností, ve kterých různé týmu vyvíjejí různé části systému. Velké systémy potřebují stabilní rámec nebo architekturu. Odpovědnost jednotlivých týmů, které pracují na částech systému, musí být předem jasně definována s ohledem na tuto architekturu. [3] [27]



Obrázek 4 – Model iteračního vývoje [3]

2.1.3 Programování řízené testy

Programování řízené testy je technika skládající se z krátkých iterací na základě automatizovaných unit testů. Namísto napsání kódu a následného testování, je napsán nejprve test.

Posloupnost kroků v procesu:

Napsání testu: Každá přidávaná vlastnost aplikace začíná napsáním testu. Tento test musí nevyhnutelně selhat, protože je napsán dříve, než je kód implementován. K napsání testu musí programátor důkladně porozumět nové funkci a jejím specifikacím.

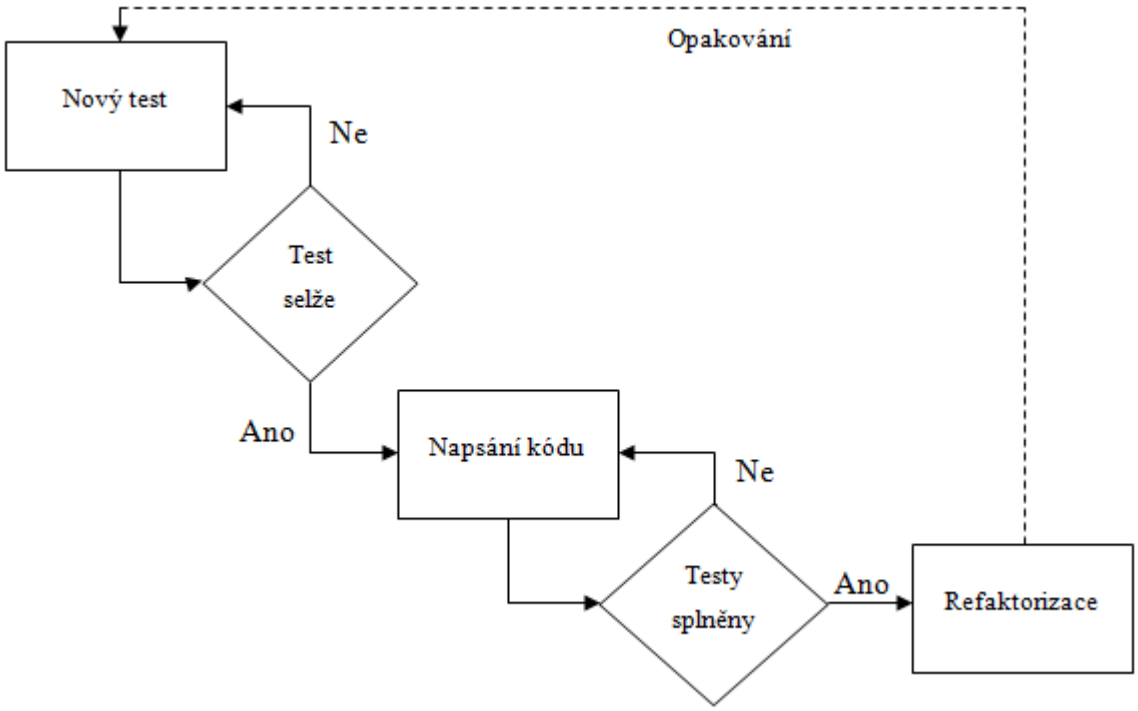
Spuštění všech testů a ujištění že nový test selže: Tímhle krokem se potvrdí, že testovací svazek pracuje správně a že nový test omylem neprojde, aniž by byl vyžadován nový kód. Kdyby nový test prošel před přidáním nové funkcionality, nemělo by ani smysl ho provádět.

Napsání kódu: V tomto kroku je cílem napsat kód tak, aby prošel testem a tím pádem splňoval svou zamýšlenou funkcionalitu. V této fázi je napsaný kód často nedokonalý a vyžaduje pozdější úpravy.

Spuštění testů: Pokud všechny budou splněny, kód splňuje všechny testované požadavky.

Refaktorizace: Tato fáze zajišťuje zpřehlednění zdrojového kódu programu bez změny jeho funkčnosti.

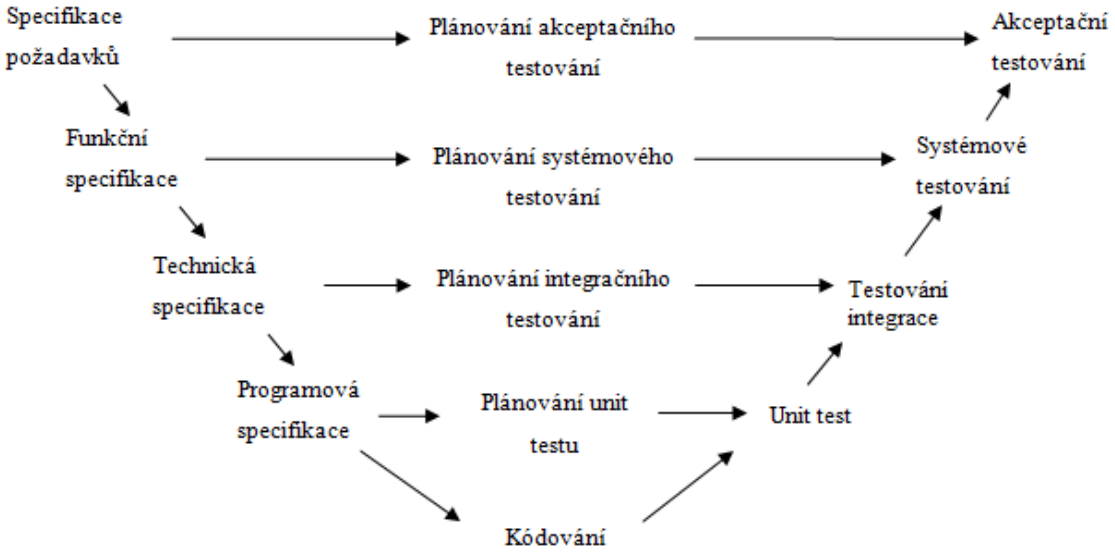
Opakování: Opět se začíná napsáním nového testu, cyklus je potom opakován k vylepšení funkcionality. [2]



Obrázek 5 – Model programování řízeného testy [2]

2.1.4 V-model

V-model je rozšíření vodopádového modelu, které umožňuje vyhodnocení produktu i v počátečních fázích. Pro každou fázi vývojového cyklu je přiřazena fáze testovací, která ověřuje vývoj. Tento model je velmi organizovaný, každá fáze začíná až po skončení té předešlé.



Obrázek 6 - Grafické znázornění V-modelu [3]

Levá část modelu je zaměřena na zpracování počátečních požadavků. S postupujícím vývojem poskytuje více technických podrobností.

Specifikace požadavků: Zachycení potřeb uživatele.

Funkční specifikace: Definice funkcí potřebných ke splnění uživatelských potřeb.

Technická specifikace: Technická specifikace funkcí definovaných v předešlé fázi.

Programová specifikace: Detailní návrh jednotlivých modulů nebo jednotek pro potřebnou funkcionalitu.

Prostřední část V-modelu ukazuje, že s návrhem testování by se mělo začít s každou vývojovou fází.

[3]

Kódování: Psaní kódu systémových modulu určených ve fázi návrhu. Nejvhodnější programovací jazyk je zvolen na základě systémových a architektonických požadavků. Kódování se provádí na základě pokynů a norem kódování. K´d prochází řadou kontrol a je optimalizován pro nejlepší výkon před finálním sestavením.

V pravé části modelu probíhá validace.

Unit test: Naplánované unit testy jsou prováděny v této fázi. Unit testování je testování na úrovni kódu a umožňuje odhalit bugy v raném stádiu, nemůže však odhalit všechny.

Testování integrace: Testování integrace je spojeno s technickou fází návrhu. Testy jsou vykonávány k vyzkoušení soužití a komunikace interních modulů v rámci systému.

Systémové testování: Systémové testování je přímo spjato s fází návrhu systému. Testy kontrolují veškerou funkcionalitu vyvíjeného systému a komunikaci s externími systémy.

Akceptační testování: Akceptační testování je spojeno s fází analýzy požadavků a obsahuje testování v uživatelském prostředí. Testy mají odhalit problémy s kompatibilitou mezi ostatními systémy uživatele. Také odhalují problémy netýkající se funkcionality, jako je výkon a odezva v uživatelském prostředí.

[32]

Výhody:

- Jednoduché a snadné použití.
- Testovací činnosti, jako je plánování a návrh testů, začíná před samotným psaním kódu. To ušetří spoustu času a dává tedy větší šanci na úspěch než v případě vodopádového modelu.
- Aktivní sledování chyb – chyby jsou nacházeny už v raných fázích.
- Zabraňuje šíření chyby do dalších fází.
- Funguje dobře pro malé projekty, kde jsou požadavky snadno srozumitelné

Nevýhody:

- Velmi tuhý a neflexibilní.
- Software je vyvíjen v průběhu implementační fáze, nejsou tedy k dispozici rané prototypy.
- Pokud v průběhu nastane změna, musí být aktualizovány dokumenty s požadavky i testy samotné.

[12]

3 PRINCIPY TESTOVÁNÍ

Testování je velmi komplexní činnost a softwarové problémy popsané v kapitole 1 ukazují, že je někdy obtížné ji udělat dobře. V průběhu let však byly vyvinuty principy, které dobrému testování napomáhají. Všechny principy nejsou zjevné, ale jejich cílem je pomoci testerům k zabránění opakování chyb.

3.1 Testování ukazuje přítomnost chyb

Spuštění testu na softwarovém systému může ukázat jen to, že se v testovaném systému vyskytuje jeden či více defektů. Testování nemůže prokázat, že je aplikace bez chyb. Při testování stránky s 10 nejhledanějšími zločinci nebyly zjištěny žádné funkční vady. Stránka přesto při spuštění selhala. V tomto případě byl problém nefunkční a absence závad nebyla dostatečným kritériem pro uvedení webových stránek do provozu. I když mohou existovat i jiné cíle, obvykle hlavním účelem testování je najít chyby. Testy by tedy měly být navrženy tak, aby se nalezlo co nejvíce závad.

3.2 Úplné testování je nemožné

Pokud jsou při testování zjištěny defekty, je pak jistě očekáváno, že další testování objeví další defekty, až by nakonec byly nalezeny všechny. Ovšem takovéto testování u komplexních systémů není možné.

Příkladem může být nepovedený start rakety Ariane 5. Nevyužitá funkcionalita modulu převzatého ze staršího typu rakety nepřímo způsobila potíže nesprávným aktualizováním proměnných. Ve starším typu rakety by tento modul fungoval správně, v prostředí novější rakety však způsobil katastrofální selhání. Kdyby všechny možné testy byly spuštěny, tato závada by byla jistě odhalena. Bohužel však nebyl věnován dostatečný čas na otestování integrace modulů.

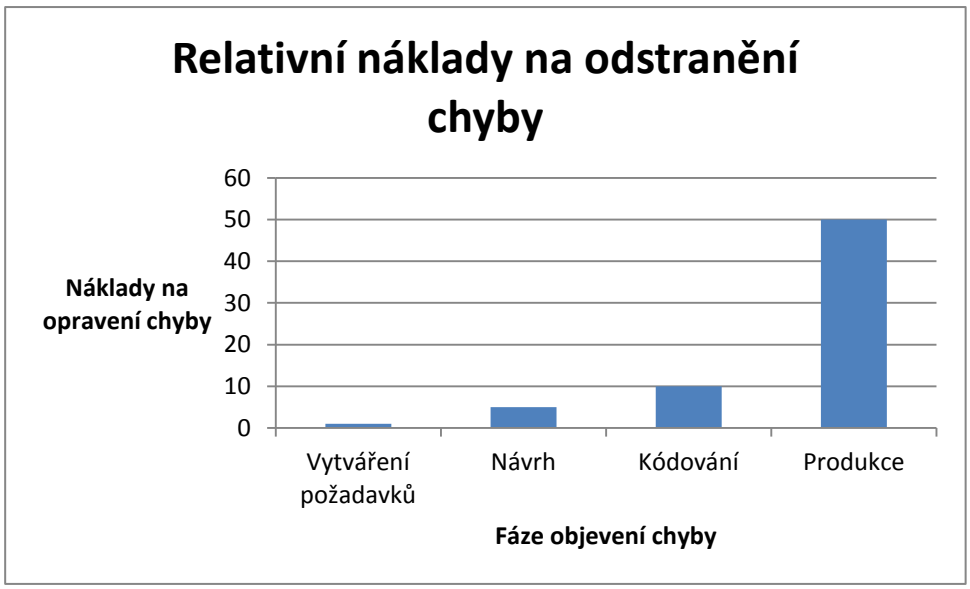
Úplné testování není možné dokonce ani u mírně složitých systémů. Pro představu, i na malé části softwaru, jako je zadání hesla o třech písmenech a použití pouze západní abecedy dostaneme 26 x 26 x 26 = 17576 možných kombinací. Na standardní klávesnici je ale více než 26 kláves. Uvažujeme-li počet možných znaků 256, potom počet kombinací je už 16777216, a to ani nebereme v potaz speciální klávesové zkratky, či vliv času, jako například vyčkání se zadáním posledního znaku do následujícího dne. Časová náročnost na otestování neroste lineárně s počtem vstupů. Čím složitější systém, tím náročnější a delší je jeho otestování. [3]

Nemá-li tedy testovaná aplikace extrémně jednoduchou logickou strukturu a omezený vstup, není možné testovat všechny možné kombinace při všech okolnostech. Z tohoto důvodu je využito hodnocení priorit a rizika pro koncentrování na nejdůležitější hlediska.

3.3 Brzké testování

Tento princip je velmi důležitý, protože při blížícím se datu vydání aplikace se může časový tlak dramaticky zvýšit. Existuje reálné nebezpečí, že čas na testování bude smrštěný. Čím dříve tedy s testováním začneme, tím více na něj budeme mít času. Pracovní produkty jsou vytvářeny v průběhu celého vývoje aplikace, a jakmile jsou připravené, můžeme je začít testovat. Vytváření akceptačních testů může začít hned po převzetí dokumentů s požadavky na vlastnosti aplikace. To může upozornit na nesrovnalosti a potenciální defekty. [3]

Nalezení a opravení defektu po dodání aplikace zákazníkovi může být až 100x dražší než jeho objevení při vytváření požadavků nebo při design fázi. Pozdější opravy vyžadují mnohem více formálních postupů a kontrolních procesů. Na toto téma bylo provedeno mnoho studií. Relativní náklady na opravení softwarové chyby v závislosti na fázi, kdy byla objevena, jsou uvedeny na *Obrázek 7*. [28]



Obrázek 7 – Relativní náklady na odstranění chyby v závislosti na fázi objevení chyby [28]

3.4 Shlukování chyb

Výskyt chyb v softwarových aplikacích není rovnoměrný. Ve velkých systémech se často v malém počtu modulů vyskytuje většina chyb. Tento fakt může být způsobem několika důvody:

- Složitost systému.
- Nestálý kód.
- Dopad změn.
- (Ne)zkušenost týmu.

Princip shlukování chyb v softwarovém testování je založen na Paretově pravidlu, také známém jako pravidlo 80-20, kde přibližně 80% závad je způsobeno 20% modulů. Nalezení chyby v jedné části aplikace, může tedy dát dobré znamení, že je pravděpodobné, že více závad bude v této části nalezeno. Je tedy vhodné na testování této části věnovat více času. Nicméně testeři by neměli ignorovat ani zbývající oblasti, kde by mohly být další defekty roztroušeny. [6]

3.5 Pesticide paradox

Tento princip nám říká, že budeme-li provádět ty samé testy stále dokola, je pravděpodobné, že žádné nové chyby těmito testy nebudou nalezeny. Vzhledem k tomu, jak se systém vyvíjí, dříve nalezené chyby budou opraveny a staré testy budou tedy už neúčinné.

Kdykoliv je defekt opraven nebo je nová funkcionalita přidána, musí být vykonáno regresní testování k ujištění, že žádné nové chyby nebyly touto změnou způsobeny. Nicméně tyto regresní testy musí být také aktualizovány, aby odrážely změny v aplikaci a byly využitelné k nacházení chyb nových. [7]

3.6 Testování je kontextově závislé

Různé metody, techniky a typy testování je třeba využít v závislosti k typu a povaze aplikace. Například program na obsluhu zdravotnického zařízení potřebuje důkladnější testování než počítačová hra, protože může přímo ohrozit zdraví či život pacienta. Je také důležité, aby testování programu zdravotnického zařízení bylo zaleženo na pokrytí hlavních rizik. Program musí splňovat nařízení zdravotnických regulačních orgánů a případně i specifických designových technik. Ze stejného důvodu, musí být populární webová stránka otestována přísnými testy výkonu a funkčnosti, aby bylo zajištěno, že její provozuschopnost nebude ovlivněna zatížením na serverech.

3.7 Klam absence chyb

Jen proto, že testování nenašlo žádné chyby v softwaru, neznamená to, že software je připraven k předání. Absence nalezených chyb nezaručuje, že požadavky zákazníka na aplikaci byly splněny. Viz podkapitola validace a verifikace. [7]

4 ÚROVNĚ TESTOVÁNÍ

Testovací fáze Životního cyklu vývoje software jsou často nazývány testovací úrovně. Název jednotlivých úrovní napovídá, na co se toto testování zaměřuje a jaké problémy nejčastěji odhaluje. Typické úrovně testování jsou:

- Unit test
- Integrační testování
- Systémové testování
- Akceptační testování
- Regresní testování

Každá z těchto úrovní obsahuje testy navržené na odhalení problémů specifických v dané fázi vývoje.

4.1 Unit test:

Unit (jednotka) je nejmenší testovatelná část software, například modul nebo funkce. Většinou je vytvářena samostatně, aby mohla být integrována v pozdějších fázích.

Unit test provádí, kromě kontroly shody se specifikacemi, ověření, že je napsaný kód spustitelný. V této úrovni testování je tedy nutný přístup ke zdrojovému kódu a provádí ho obvykle programátor s využitím nástrojů pro debugging. Chyby nalezené v této fázi se často nezaznamenávají.

Výhodou této úrovně testování je, že nachází chyby velmi brzy a tedy snižuje náklady na jejich opravu. Za nevýhodu může být naopak považován fakt, že programátor testuje svůj vlastní kód, a může se stát, že test navrhne tak, aby vyhovoval napsanému kódu a ne nutně specifikacím. Tato nevýhoda se dá zmírnit využitím dvou programátorů, kdy napsaný kód není testován autorem. [3][19]

4.2 Integrační testování

Jakmile jsou všechny jednotky napsány, další etapa spočívá v sestavení systému. Toto sestavení se nazývá integrace a zahrnuje budování celku z menších částí. Účelem testování integrace je odhalit vady v rozhraní a interakci jednotlivých integrovaných jednotek nebo systémů.

Před započetím plánování integračních testů, je nutné zvolit integrační strategii. Jedná se o rozhodování o tom, jak tento systém bude před testováním sestaven. Nejčastěji zmiňované integrační strategie jsou;

4.2.1 Integrace „Velký třesk“

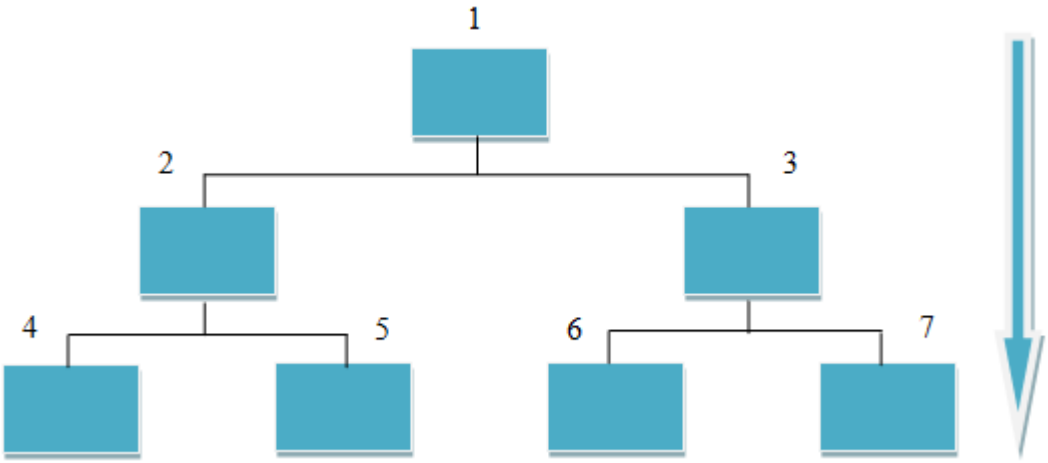
Integrace Velký třesk, z anglického *Big-bang integration*, spočívá v propojení všech jednotek najednou. Složí se tak kompletní systém. Testování takovéhoto systému přináší potíže v izolování nalezených chyb, protože pozornost není věnována ověření rozhraní mezi jednotlivými jednotkami.

Tento typ integrační strategie je obecně považován za špatný výběr. Přináší problémy, kterou mohou být zjištěny až v pozdějších fázích projektu, kde jsou dražší opravit.

4.2.2 Integrace „Shora-dolů“

Integrace Shora-dolů, z anglického *Top-down integration*, spočívá v sestavení systému v několika fázích, počínaje komponenty, které volají další. Komponenty, které jsou volány, jsou umístěny pod těmi volajícími. Integrace Shora dolů umožňuje testerovi vyzkoušet rozhraní jednotek, počínaje těmi na vrcholu.

Tato struktura může být zobrazena schématem. Na Obrázek 8 je zobrazena integrace, kdy komponenta 1 volá komponenty 2 a 3, je tedy umístěn nad nimi. Komponenty 2 a 3 mohou volat komponenty 4 a 5, resp. 6 a 7, a proto jsou umístěny nad nimi. [3][19]

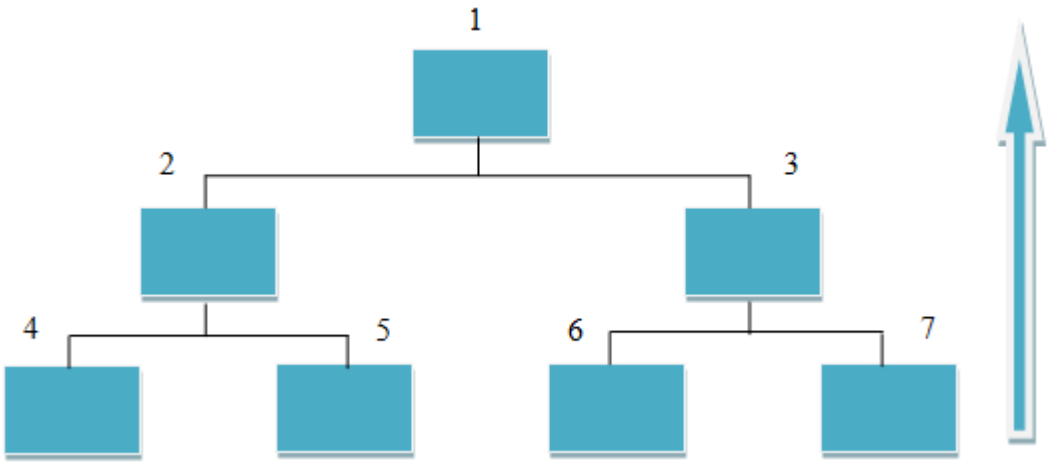


Obrázek 8 – Schéma integrace typu Shora-dolů [3]

Integrace typu Shora-dolů vyžaduje testování interakce jednotlivých komponent při sestavení. Níže postavené jednotky nemusí být v době testování dokončeny, nebo integrovány. Mohou být dočasně nahrazeny náhradou, jakousi kostrou budoucí jednotky. Tato náhrada se nazývá „*stub*“. Stub je pasivní prvek, který je volán ostatními komponenty. Na uvedeném schématu mohou tyto náhražky zastoupit komponenty 6 a 7 při testování komponenty 3.

4.2.3 Integrace „Zdola-nahoru“

Integrace Zdola-nahoru, z anglického *Bottom-up integration*, je opak integrace Shora-dolů. Při sestavení se nejprve integrují prvky na nižší úrovni. Z Obrázek 9 by to byla integrace nejprve komponent 4-7 a až poté komponent 2 a 3. V tomto případě by tedy komponenty, které aktivně volají komponenty další, nemuseli být ještě integrovány. Musejí tedy být, podobně jako v integraci Shora-dolů, nahrazeny speciálně napsanými komponentami. Tyto náhražky se nazývají „*drivers*“. Drivers jsou aktivní prvky, které ovládají komponenty na nižší úrovni. [3][19]



Obrázek 9 – Schéma integrace Zdola-nahoru [3]

4.3 Systémové testování

V této fázi již jsou komponenty napsány a integrovány. Nyní je tedy vytvořen kompletní systém, který je podroben důkladnému testování, aby se určilo, zda splňuje stanovené požadavky. U komplexních aplikací může být toto testování rozděleno do několika fází, kde jsou komponenty integrovány do subsystémů, které jsou individuálně otestovány před finální integrací. Toto testování už převážně vykonává specializovaný tým.

- Systémové testování je první fází, kde je aplikace testována jako celek.
- Aplikace je otestována důkladně k ověření, že splňuje funkční i nefunkční požadavky.
- Aplikace je testována v prostředí, které je velmi blízké produkčnímu, kde bude aplikace nasazena. [27][31]

4.4 Akceptační testování

Akceptační testování je poslední etapa v procesu testování před uvedením systému do provozu. Systém již není testován se simulovanými daty, ale s daty poskytnutými od zákazníka. Akceptační testování může odhalit chyby a opomenutí v systémových požadavcích, protože reálná data působí na systém jinak, než ta simulovaná. Toto testování může také odhalit, že systémové chování nevyhovuje uživatelským potřebám anebo, že jeho výkon je nepřijatelný.

Akceptační testování není určeno pouze na odhalení překlepů, kosmetických chyb, nebo mezer v rozhraní, ale také poukázat na chyby, které mohou vést k pádům aplikace anebo jiným, závažným, chybám. [27][31]

4.5 Regresní testování

Kdykoliv je provedena změna v softwarové aplikaci, je docela možné, že jiné oblasti v rámci aplikace byly touto změnou zasaženy. Chceme-li ověřit, že oprava chyby nevedla k porušení pravidel či požadavků na danou aplikaci, provádíme regresní testy. Záměrem regresního testování tedy je ověřit, že opravení jednoho bugu nezpůsobilo bug v jiném místě aplikace. Regresní testování je důležité z těchto důvodů:

- Testování změny k ověření, že změna nezpůsobila nežádoucí jev v jiné části aplikace.
- Zvyšuje pokrytí testy, bez ohrožení časových uzávěrek.
- Zvyšuje rychlo uvedení produktu na trh.

[31]

5 METODY A TYPY TESTOVÁNÍ

5.1 Metody testování

K dosažení větší efektivity testování, jsou testy psány využitím různých metod. Tímto se dosáhne toho, že jsou zvoleny testovací podmínky s největší pravděpodobností odhalení chyby v softwaru. Testeři tak nemusí hádat co a jak testovat, ale mohou testy vytvářet systematicky. Testovací metody se dělí do několika kategorií, mezi ty hlavní patří například black box a white box techniky.

[14]

5.1.1 Black box

Black box je technika testování bez znalostí vnitřního fungování aplikace. Tester nezná architekturu systému a nemá přístup ke zdrojovému kódu, pouze ví, co by měla aplikace dělat, ale nemůže nahlédnout dovnitř. Typicky, při provádění testů za pomocí black box metody, tester zadává nějaké vstupní hodnoty a pozoruje hodnoty výstupní, neví však jak, nebo proč tyto hodnoty vznikly.

[22][34]

Tabulka 1 – Srovnání výhod a nevýhod black box metody [34]

Výhody:	Nevýhody:
Vhodné a efektivní pro velké kódové segmenty.	Omezené pokrytí, protože se provádí pouze vybrané testy.
Není nutný přístup ke kódu.	Neefektivní testování, protože má tester pouze omezené znalosti o aplikaci.
Jasně odděluje perspektivu uživatele a vývojáře.	Tester nemůže zaměřit testování na specifické části kódu nebo oblasti náchylné k výskytu chyb.
Zkušení testeři mohou otestovat aplikaci, bez dalších znalostí implementace, programovacího jazyka nebo operačních systémů.	Návrh testů je poměrně složitý.

5.1.2 White box

White box je technika testování se znalostí vnitřní struktury a architektury. Tester má v tomto případě přístup ke zdrojovému kódu, kde může získat nějakou nápovědu, které vstupy by mohly potenciálně způsobovat chyby. [22][34]

Tabulka 2 - Srovnání výhod a nevýhod white box metody [34]

Výhody:	Nevýhody:
Se znalostí zdrojového kódu je mnohem snazší identifikovat data umožňující efektivnější testování.	Zkušený tester je vyžadován při white box testování, zvyšují se tedy i náklady.
Pomáhá s optimalizací kódu.	Někdy je nemožné prozkoumat celý kód důkladně a odhalit tak všechny potencionální skryté chyby.
Mohou být odstraněny části kódu k odhalení skrytých defektů.	
Se znalostí vnitřní struktury je také zvýšeno testovací pokrytí.	Jsou vyžadovány specializované nástroje jako analyzéry kódu a debuggující nástroje.

5.2 Typy testování

V předchozí kapitole byly určeny specifické cíle testování pro každou úroveň. Typy testování jsou kategorie, za pomocí kterých se těchto cílů dá dosáhnout.

Patří sem mimo jiné:

- Funkční testování
- Nefunkční testování
- Testování struktury
- Testování po změně kódu

[24]

5.2.1 Funkční testování

Funkční testování se zaměřuje na specifické funkce systému a ověřuje, zda jsou splněny specifikace. Jednotlivé funkce aplikace jsou prováděny a jejich výstup je zkoumán. [3]

V případě webových aplikací, je funkční testování zaměřeno na odkazy, spojení s databází, získávání informací od uživatele, testování Cookies, atd.

Test odkazů na stránce: Testují se odkazy odkazující na vnitřní i vnější stránky v rámci dané domény a hledají se nefunkční linky a chyby v jejich propojení.

Test spojení s databází: Konzistence dat je velmi důležitá pro webové aplikace. Proto se kontroluje integrita dat a chyby v databázi při vykonávání jakékoliv činnosti související s prací s databází. Provádí se kontrola databázových dotazů, načítání dat a jejich úprava. Může být také testována zátěž a výkon databáze.

Test formulářů: Formuláře, jako jeden z prostředků získávání informací od uživatele, musí být otestovány, zda správně přenáší informace a ukládá je do databáze. Dále se provádí kontrola, jestli formulář správně validuje vstup od uživatele i jestli jeho funkce, jako vymazání textu, fungují správně.

Test Cookies: Cookies jsou malé soubory ukládané do počítače uživatele sloužící k udržení relace, převážně relace přihlášení. Testuje se chování aplikace s povolenými/nepovolenými Cookies.

[25]

Skripty a knihovny: Testováním skriptů a knihoven ověřujeme, že jsou kompatibilní se všemi různými prohlížeči a jejich výkon i doba provádění je optimální.

Multimédia: Testuje se přehrávání, zobrazení a interakce nejrůznějších multimediálních prvků jako je video, zvuk, animace a interaktivní komponentů. Tyto prvky by měly běžet dle očekávání a nezpůsobovat chyby, nebo zpomalení ve zbytku aplikace při načítání nebo běhu.

[8]

5.2.2 Nefunkční testování

Při nefunkčním testování se testují aspekty chování aplikace. Ověřují se požadavky aplikace, které nejsou zaměřeny na funkce aplikace, ale jsou stejně důležité. Může se jednat například o testování aplikace v zátěži, její bezpečnost, nebo uživatelské rozhraní.

[31]

5.2.2.1 Testování výkonu

Testování výkonu je zaměřeno více na odhalení překážek nebo problémů s výkonem, než na hledání chyb v softwaru. Ke snižování výkonu aplikace přispívají různé příčiny, mezi něž patří například:

- Zpoždění sítě.
- Zpracování transakcí v databázi.
- Zatížení serveru.
- Zobrazování dat.

Testování výkonu je považováno za velmi důležité a mělo by být povinnou součástí testování jakékoliv aplikace, co se týče testování parametrů jako stabilita, škálovatelnost, rychlost odezvy, atd. Testování může být kvalitativní nebo kvantitativní. Pod testování výkonu také patří dílčí testování zátěže a stres testování.

Testování zátěže

Testování zátěže je proces ověřování chování aplikace s použitím maximálního zatížení v oblasti přístupu a manipulace s daty. Tento typ testování určuje maximální kapacitu software a jeho chování ve špičce. K testování zátěže se často používají automatizované nástroje generující virtuální uživatele, kteří provádějí daný skript. Jejich počet může být průběžně měněn na základě požadavků.

Stres testování

Hlavním smyslem stres testování je odhalení bodu, kdy aplikace přestane pracovat. K určení tohoto tzv. bodu zlomu se aplikace testuje za abnormálních podmínek. Tyto podmínky mohou být způsobeny odebráním prostředků nutných pro chod aplikace anebo použití zatížení za hranicí schopností softwaru. Toho může být dosaženo například:

- Vypínání a zapínání databáze.
- Blokování síťových portů.
- Spuštění různých procesů, které spotřebovávají zdroje, ať už CPU, paměť, nebo server. [31]

5.2.2.2 Testování použitelnosti

Testování použitelnosti se vztahuje k vyhodnocení a ověření, že je práce s aplikací efektivní. Je to black box technika a slouží k identifikaci jakékoliv chyby nebo návrhu zlepšení tím, že se sleduje uživatel při práci s aplikací.

Testování použitelnosti hraje klíčovou roli při návrhu aplikace tak, aby manuální úlohy byly co nejvíce ulehčené. Rozhraní aplikace by mělo splňovat určité standardy a celosvětově uznávané konvence, například při využití barev, červená naznačuje ukončení

procesu, zelená naopak jeho spuštění. Jejich zaměnění by vedlo ke zmatení uživatele a v důsledku k neefektivní práci s aplikací.

[21][31]

Použitelnost je součástí „užitečnosti“ aplikace a skládá se z:

- Naučitelnost: Jak snadné pro uživatele je, provést základní úlohy při prvním styku s designem.
- Účinnost: Jak rychle se dají splnit úlohy, poté co se uživatel seznámil s designem.
- Zapamatovatelnost: Když se uživatel vrátí k aplikaci po delší době neužívání, jak snadno obnoví znalosti.
- Chyby: Kolik chyb uživatel dělá, jak závažné jsou a jak snadno se překonávají.
- Spokojenost: Jak příjemné je používat daný design.

[18]

Při testování použitelnosti webových stránek by měla být pozornost věnována mimo jiné těmto oblastem:

Navigace: Při testu navigace se sleduje, jak uživatel prochází stránky, využívá různé kontrolní prvky jako tlačítka, nebo jak využívá odkazy na další stránky. Tím se ověřuje, zda jsou instrukce jasné, správné a slouží svému záměru. Hlavní menu by mělo být konzistentní a dostupné na každé stránce.

Kontrola obsahu: Při testu obsahu se ověřuje, zda je logicky uspořádán a lehce srozumitelný. Hledají se také překlepy, chyby v textu a jiné prohřešky proti základním standardům.

Uživatelská nápověda: Pokud stránka obsahuje navigační prvky, vyhledávání nebo nápovědu, tyto položky by také měly být přezkoumány.

[25]

5.2.2.3 Testování bezpečnosti

Zabezpečení aplikace zahrnuje opatření přijatá v průběhu životního cyklu kódu. Testování bezpečnosti má tedy za cíl zjistit jakékoliv nedostatky a mezery v bezpečnosti a zranitelná místa aplikace.

[31]

Jedny z nejvíce využívaných bezpečnostních slabin webových aplikací je cross-site scripting (XSS) a SQL injection.

Cross-site scripting: XSS je výsledkem vnoření škodlivého skriptu do webové stránky. Tento skript může dovolit třetím osobám sbírat informace o uživatelích a jejich přístupových informací.

SQL injection: S rostoucí závislostí webových stránek na databázích, roste i využívání této slabiny. Podstata SQL injection je ve spuštění databázového příkazu v místě, ve kterém to nebylo zamýšleno vývojářem, za účelem překonání autentizace uživatele, nebo k manipulaci s daty.

[20]

5.2.3 Manuální testování

Tento typ testování zahrnuje testování softwarové aplikace manuálně, tzn. bez užití automatizovaného nástroje nebo skriptu. Při testování bere tester na sebe roli koncového uživatele ve snaze odhalit nějaké nežádoucí chování nebo přímo bug v aplikaci. [35] Jednotlivé úrovně manuálního testování byly pokryty v kapitole 4.

5.2.3.1 Výhody manuálního testování

Krátkodobé náklady jsou nižší: Nástroje automatického testování bývají drahé. S manuálním testováním není zapotřebí vynakládat značné částky na jejich nákup.

Větší pravděpodobnost odhalení problémů reálného uživatele: Manuální testování umožňuje testování aplikace za podmínek lépe odrážejících ty, za kterých bude běžet po vydání. Je tedy více pravděpodobné odhalení chyb při zacházení s aplikací určitým způsobem.

Manuální testování je flexibilní: Změny v návrhu testu se do automatického testování promítají složitěji a stojí to více času. Při manuálním testování se změny dají otestovat rychleji a jednodušeji.

5.2.3.2 Nevýhody manuálního testování

Některé úlohy se obtížněji provádějí manuálně: Některé kroky se snáze provádějí za pomocí automatických testů. Například simulování více uživatelů anebo testování rozhraní na nízké úrovni.

Testování může být nudné a stále se opakující: Při vyplňování stále stejných dat a formulářů může tester ztratit soustředění a stát se více náchylným na přehlédnutí chyby.

Manuální testy nelze znovu použít: Automatizované testy je možné spustit ihned, jakmile je to potřeba. Při manuálním testování, pokud je provedena nějaká změna v aplikaci, musí se test provést znovu. To je značná ztráta času.

[1]

5.2.4 Automatické testování

Při automatickém testování tester vytváří automatické skripty a využívá různé softwarové nástroje pro testování. Proces automatického testování zahrnuje automatizaci nějakého manuálního testovacího procesu. Automatizace testů umožňuje jejich opětovné spuštění a rychlejší provedení.

[31]

5.2.4.1 Výhody automatického testování

Provedení testů rychle a efektivně: Přestože počáteční vytvoření automatického testu může zabrat notnou chvíli, jakmile je hotový, jeho provedení je rychlejší a efektivnější.

Nákladově výhodné: V krátkodobém hledisku jsou nástroje automatického testování nákladné, dlouhodobě však náklady šetří. Nejenže ve stejném čase toho stihnou více, ale také nachází bugy dříve, čímž umožňují rychlejší reakci.

Zpříjemňují práci: Vyplňovat stále dokola ty samé formuláře může být pro testera frustrující a nudné. Automatizovaný test tento problém řeší automatickým vyplňováním.

Každý může vidět výsledky: Po přihlášení do testovacího systému nebo nástroje automatického testování může každý z členů vidět výsledky testování. To usnadňuje spolupráci týmu a zlepšuje finální produkt.

5.2.4.2 Nevýhody automatického testování

Nástroje mohou být drahé: Počáteční investice do nákupu nástrojů automatického testování může být vysoká. Je tedy důležité vybrat správný nástroj.

Testování může být stále velmi časově náročné: Přestože se čas provádění testů zkrátí, provedení většího počtu testů stále nějaký čas trvá. Pro vytvoření automatických testů je také zapotřebí značná část času.

Omezení nástrojů: Přestože automatické testy dokážou odhalit většinu bugů v systému, mají své omezení. Nemohou například testovat vizuální aspekty jako barva obrázku anebo velikost písma. Nežádoucí změny těchto parametrů mohou být odhaleny pouze manuálním testováním. [1]

II. PRAKTICKÁ ČÁST

6 TESTOVACÍ NÁSTROJE

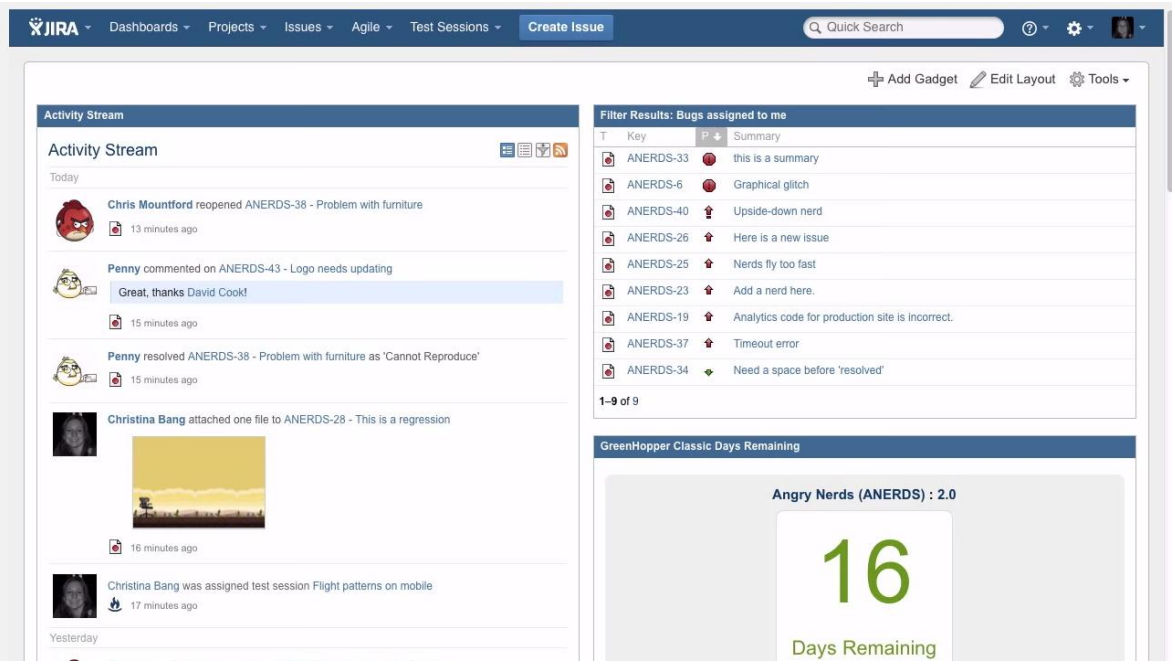
Jak již bylo zmíněno, testování webových aplikací, i testování softwarových aplikací obecně, obsahuje mnoho úkolů a činností, které je třeba vykonat během procesu testování. Kromě toho, vykonání dalších úloh může být vyžadováno před započetím samotného testování. Pro snazší kontrolu a provádění testů byly vyvinuty nejrůznější nástroje.

6.1 JIRA

JIRA je nástroj umožňující sledování problémů (issues), vyvinutý firmou Atlassian. Je to komerční softwarový produkt, který může být licencován na provoz na místě nebo jako hostovaná aplikace. Umožňuje sledování chyb, problémů a defektů, a také poskytuje funkce pro správu projektu, např. sledování času.

6.1.1 Základní prvky programu

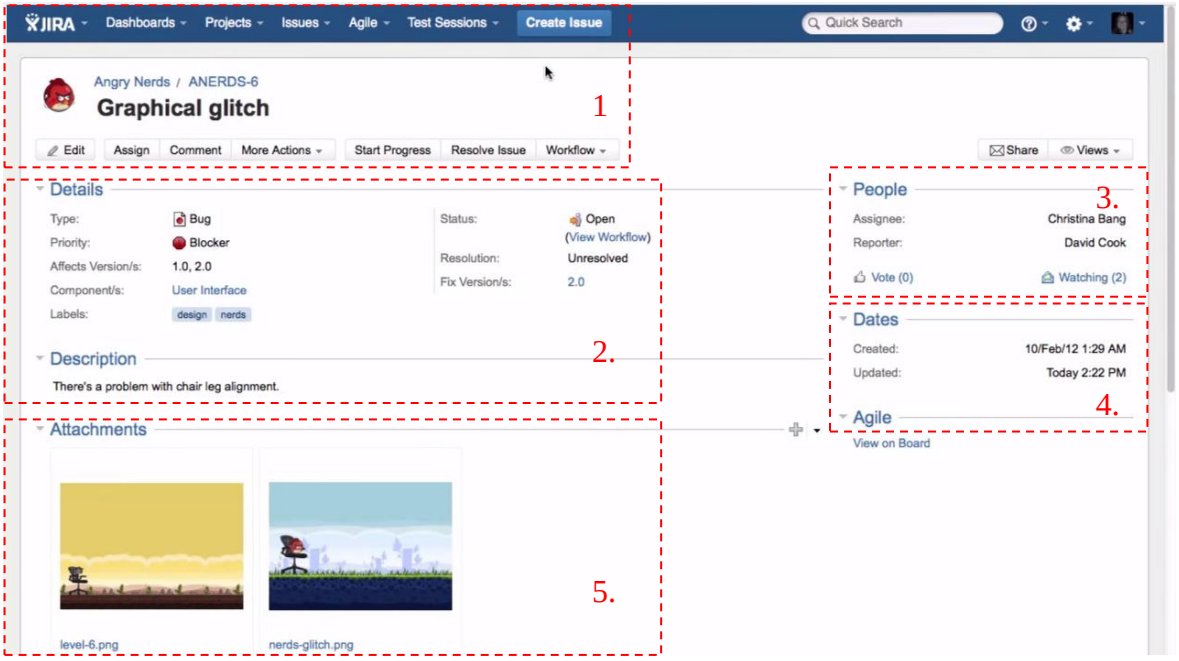
Dashboard je hlavní projektová deska. Zobrazení se dá filtrovat tak, aby každý uživatel viděl to podstatné na projektu jako první. Zobrazení lze nastavit například tak, aby jako první bylo vidět poslední aktivita na projektu, bugy přiřazené k uživateli a zbývající dny do dokončení projektu.



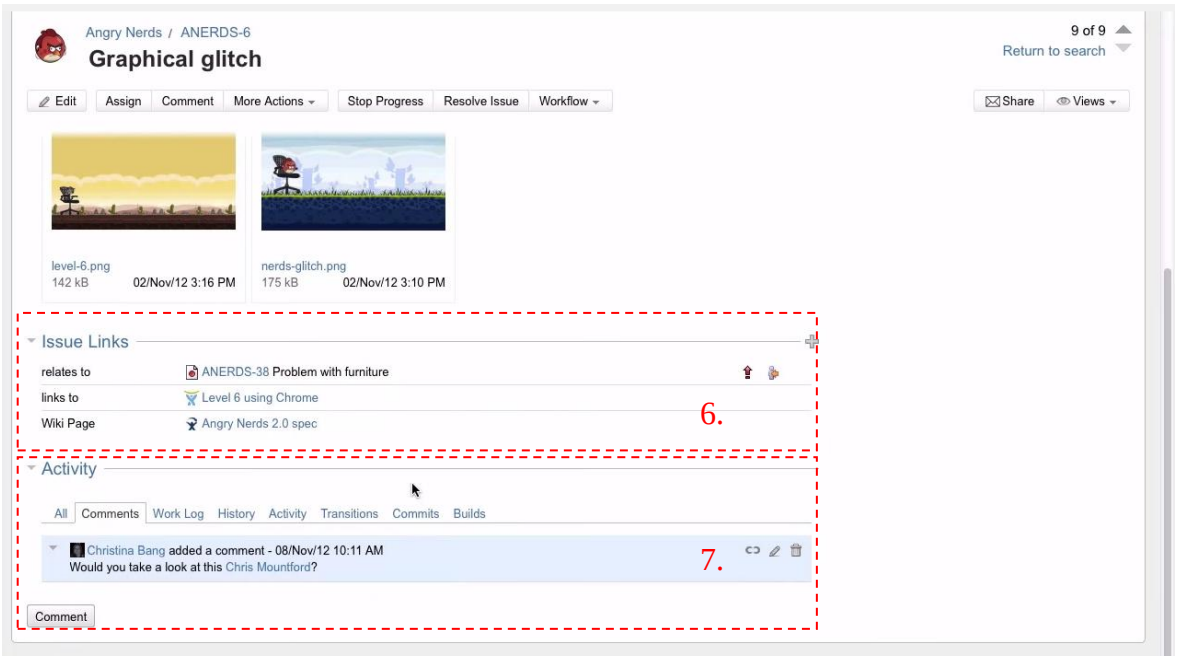
Obrázek 10 – JIRA dashboard

Issue je základní prvek systému. Obsahuje veškeré informace ohledně problému, funkcionality, nebo vlastnosti, která se právě řeší.

Informace obsažené v každém issue jsou rozděleny do několika sekcí.



Obrázek 11 – JIRA issue



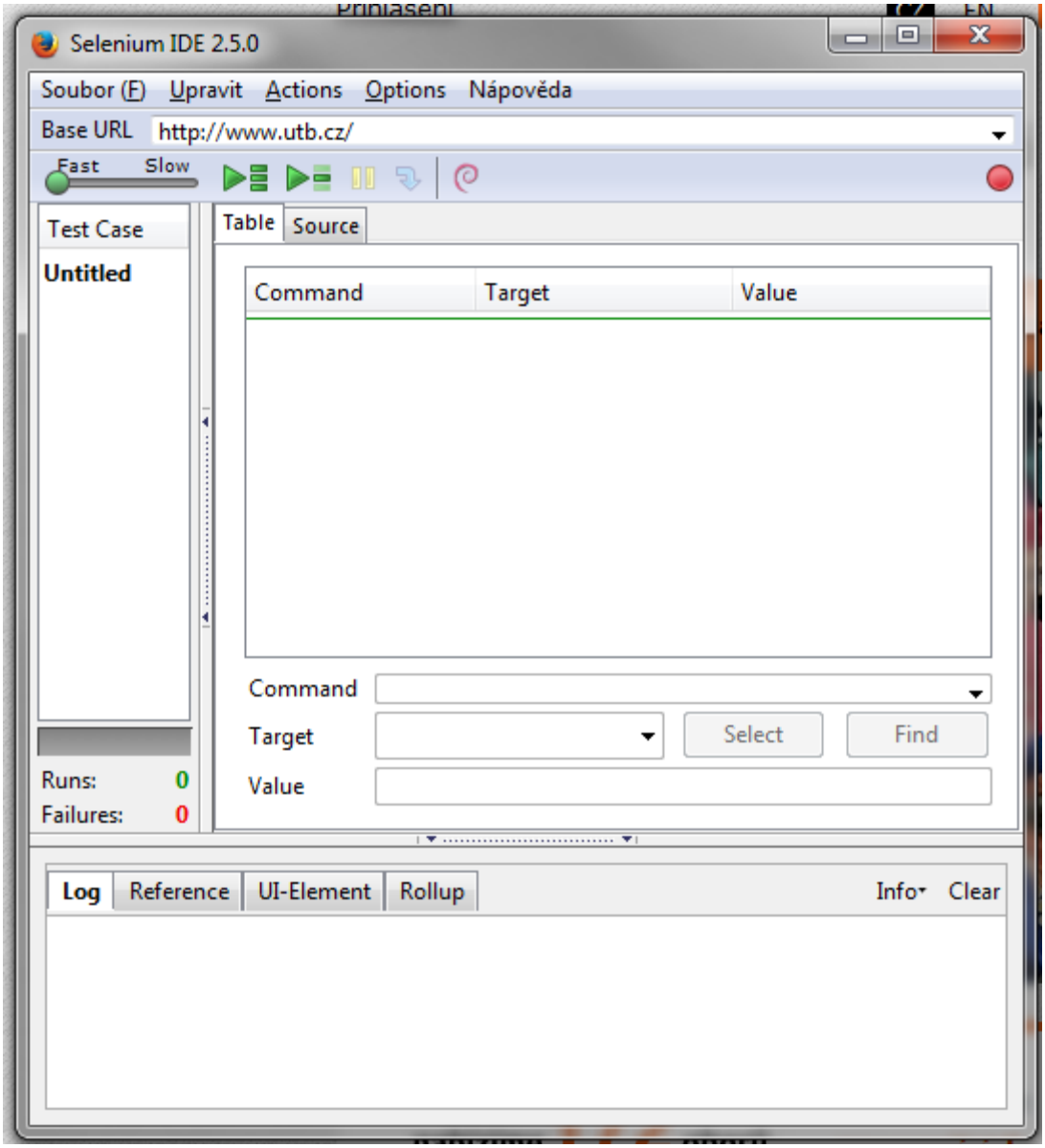
Obrázek 12 – JIRA issue – spodní sekce

1. Název issue a kontrolní funkce
2. Detaily jako typ, status, priorita, verze a popis
3. Lidé – uživatel kdo issue vytvořil a komu je přiřazen
4. Data – datum vytvoření a poslední aktualizace
5. Přílohy – obrázky, dokumenty a jiné
6. Linky – zde mohou být uvedeny odkazy na relevantní issue
7. Aktivita – přehled komentářů a provedených změn

6.2 Selenium IDE

Selenium IDE (Integrované vývojové prostředí) je přenosný softwarový rámec pro testování webových aplikací. Je to snadno použitelný plug-in do prohlížeče Firefox poskytující nástroj pro záznam a přehrávání automatických testů bez znalosti skriptovacího jazyka.

6.2.1 Grafické rozhraní



Obrázek 13 – Grafické rozhraní pluginu Selenium IDE

Panel nástrojů:

Obrázek 14 – Panel nástrojů Selenium IDE

Panel nástrojů nám umožňuje oválení nástroje. Posuvník umožňuje uživateli ovládat rychlost vykonávání testů. Další tlačítka umožňují výběr přehrání mezi jedním, nebo všemi nahranými testy. Dále můžeme vykonávání testu zastavit a znovu spustit tlačítkem Pauza, spouštět kód krok po kroku. Další tlačítko je pro speciální funkci, která umožňuje opakované příkazy shlukovat do jedné akce. Poslední tlačítko slouží pro záznam

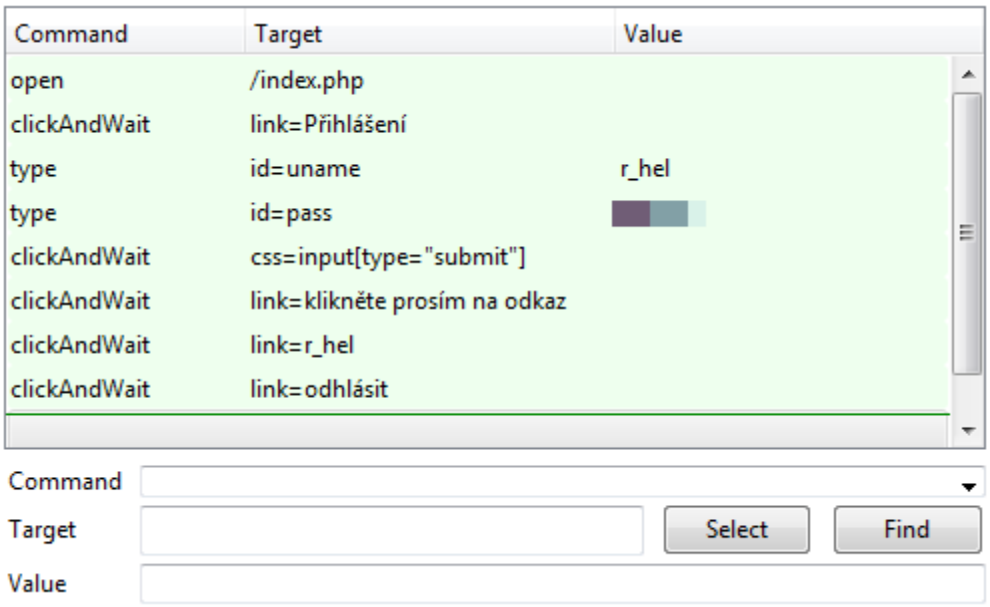
kroků uživatele v prohlížeči. Tato funkce bude využita pro názornou ukázku práce s programem.

6.2.2 Záznam a spuštění testovacího případu

Byl vytvořen jednoduchý automatický test, ověřující možnost přihlášení na stránky www.utb.cz a možnosti navigace na profil uživatele. Kroky se skládaly z:

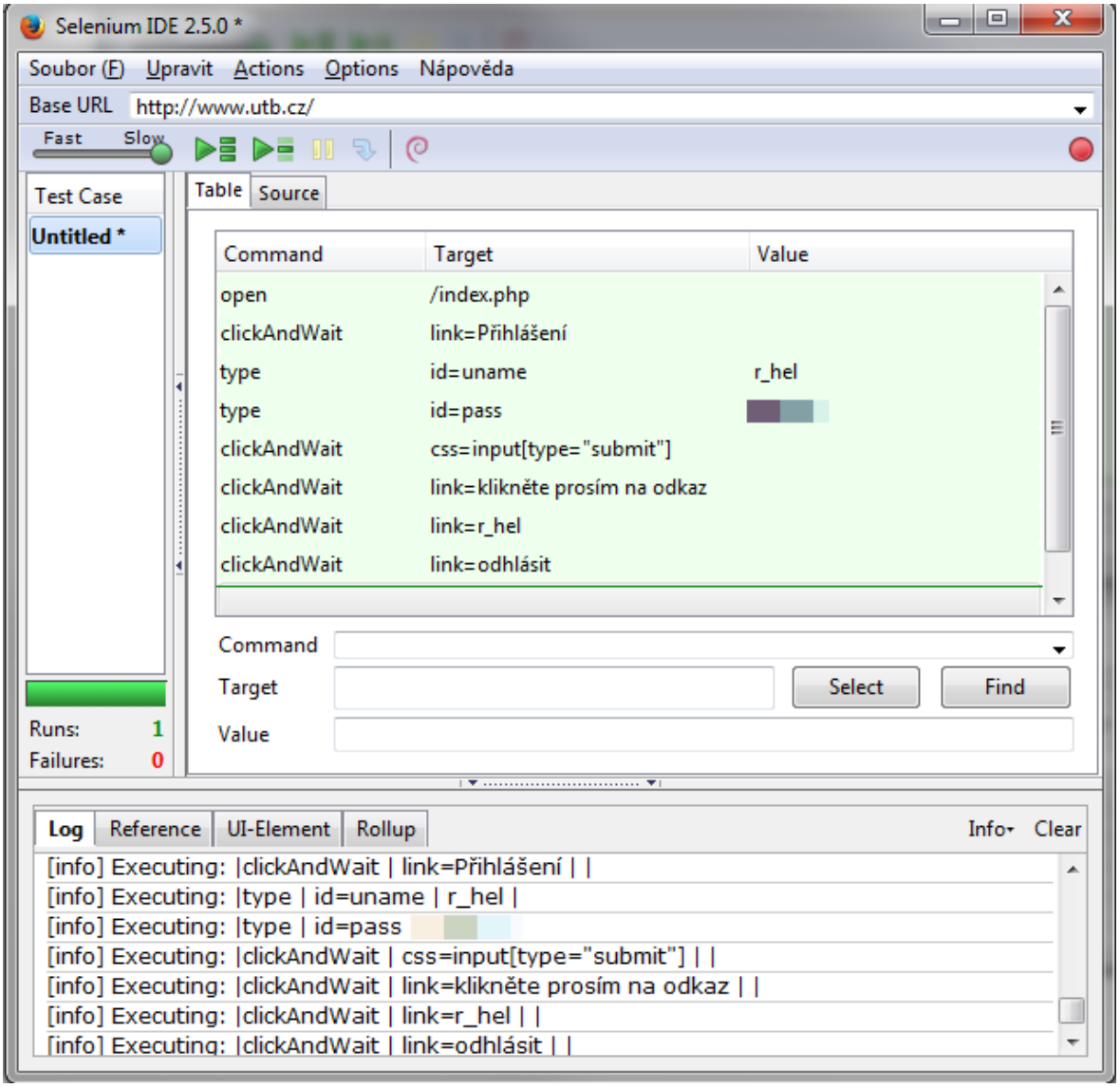
1. Navigace na stránky **http://www.utb.cz**
2. Kliknutí na odkaz přihlášení
3. Zadání uživatelského jména a hesla
4. Navigace na uživatelský profil
5. Odhlášení

Nástroj Selenium IDE na základě tohoto postupu vytvořil automatický skript s těmito parametry:



Obrázek 15 – Parametry vytvořeného skriptu

Následně byl skript spuštěn a automatický test proběhl úspěšně:



Obrázek 16 – Výsledek spuštění automatického skriptu

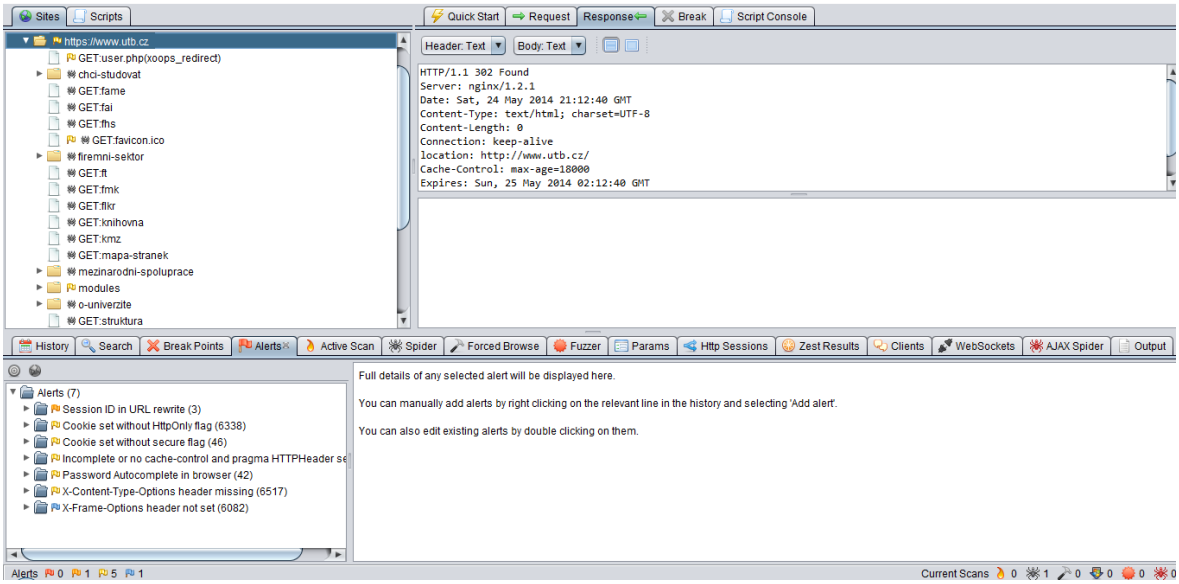
6.3 OWASP Zed Attack Proxy

Zed Attack Proxy (ZAP) je snadno použitelný nástroj pro testování proniknutí a hledání zranitelných míst webových aplikací. Je určen pro širokou škálu lidí, ať už s bohatými zkušenostmi o bezpečnosti, nebo pro teprve začínající s testováním proniknutí.

ZAP poskytuje automatizované skenery a další nástroje, které umožňují ruční hledaní zranitelných míst monitorováním komunikace mezi prohlížečem a serverem.

6.3.1 Obsluha programu

Pro zobrazení práce s nástrojem byl spuštěn test stránek www.utb.cz. Nástroj ZAP prozkoumal více jak 40000 domén a sub-domén a nalezl celkem 7 upozornění na možné slabiny. Jako nejrizikovější program vyhodnotil používání jména uživatele v ID relace. Toto jméno může být odhaleno v odkazové hlavičce. Kromě toho, ID relace může být uloženo v historii prohlížeče nebo v záznamu serveru. Jako vyřešení této situace nástroj nabídl využití cookies.



Obrázek 17 – Výsledky skenování programu ZAP

Obrázek 18 – Upozornění na nedostatky v zabezpečení

6.4 Broken Link checker

Broken Link je nástroj prohledávající webovou stránku za účelem odhalení neplatných hypertextových odkazů. Neplatné odkazy nejen objeví, ale také zobrazí, v jakém místě HTML kódu se tyto odkazy nacházejí, takže jejich odstranění, nebo náprava, je mnohem jednodušší.

6.4.1 Výhody a nevýhody

Výhody:

- Nástroj je dostupný online a je tedy přístupný kdykoliv a odkudkoliv bez závislosti na operačním systému.
- Bezplatná verze umožňuje prohledání až 3000 webových stránek.
- Nástroj zobrazí umístění neplatného hypertextového odkazu v HTML kódu.

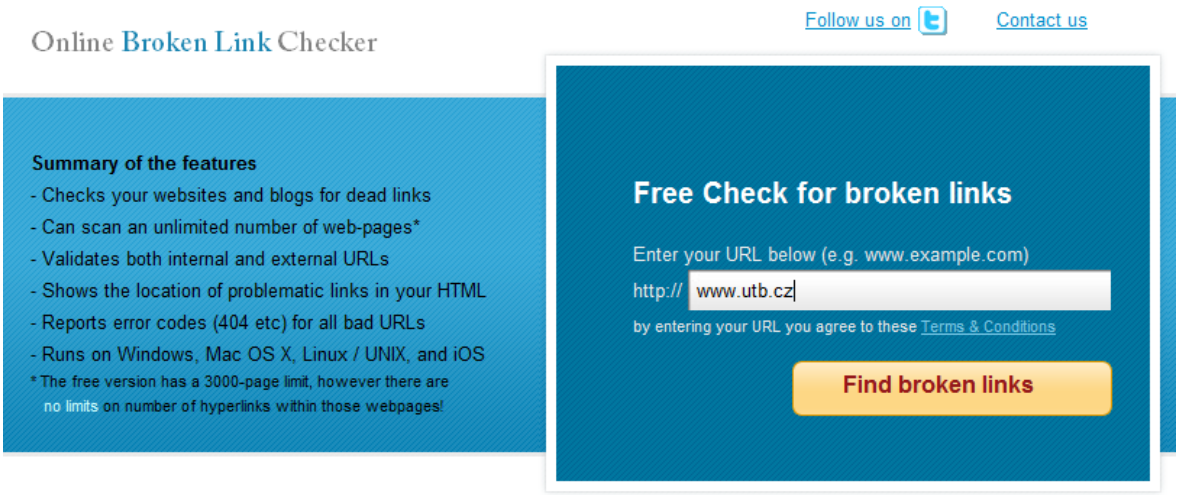
Nevýhody:

- Omezení počtu skenovaných stránek v bezplatném režimu.
- Nutné připojení k internetu během provádění testu.

6.4.2 Obsluha programu

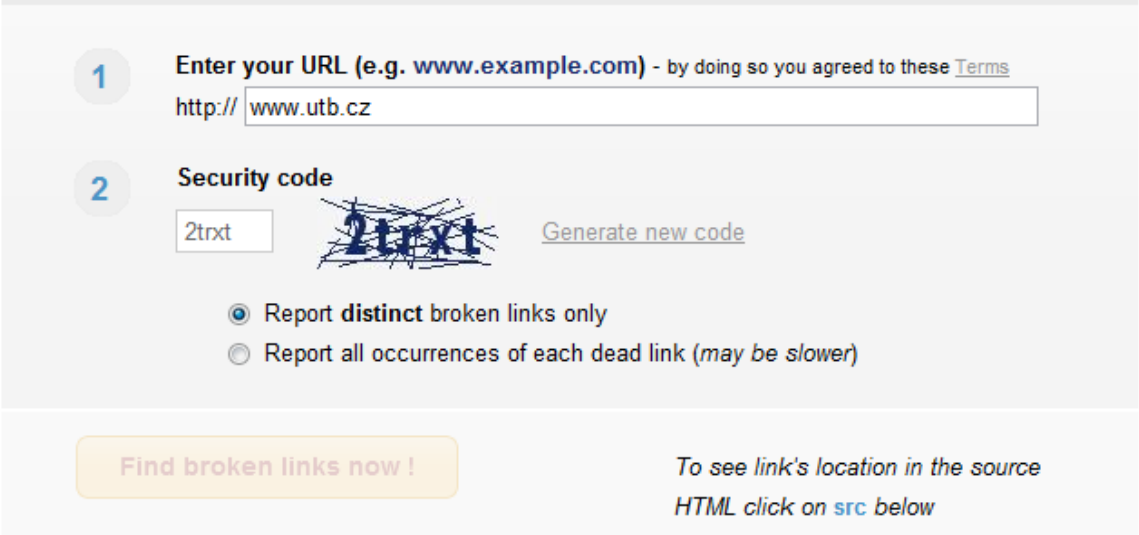
Nástroj je dostupný na adrese: **http://www.brokenlinkcheck.com/**

1. Zadání URL adresy do vyhledávače nástroje



Obrázek 19 – Zadání adresy do vyhledávače Broken Link

2. Ověření bezpečnostního kódu



Obrázek 20 – Zadání bezpečnostního kódu na stránce nástroje Broken Link

3. Zobrazení výsledků

#	Broken link (you can scroll this field left-right)	Page where found	Server response
1	http://www.msmt.cz/strukturalni-fondy/op-vavpi	url src	404
2	http://www.strukturalni-fondy.cz/getdoc/977e2e36-937e-4432-afe7-165af >>	url src	404
3	http://www.utb.cz/modules/marwel/studium@flkr.utb.cz	url src	404
4	http://www.utb.cz/struktura/pohovor-nanecisto-nabidka-pozic >>	url src	404
5	http://www.adobe.com/products/acrobat/main.html	url src	404
6	http://www.utb.cz/mezinarodni-spoluprace/pracovni-staz	url src	404
7	http://www.utb.cz/modules/marwel/halamik@rektorat.utb.cz	url src	404
8	http://www.utb.cz/modules/marwel/zahorovska@rektorat.utb.cz%20 >>	url src	404
9	http://www.utb.cz/modules/marwel/zahorovska@rektorat.utb.cz >>	url src	404
10	http://www.utb.cz/fame/struktura/caev	url src	404
11	http://www.utb.cz/mezinarodni-spoluprace/norsko	url src	404
12	http://www.yafa.am/en/	url src	bad host
13	http://www.sdu.edu.cn/english/	url src	404
14	http://www.unigambia.gm	url src	bad host
15	http://www.wsb.net.pl/uczelnia/english	url src	404
16	http://www.bg.ac.rs/en_index.php	url src	404
17	http://www.lnu.edu.ua/Department/index.htm	url src	404
18	http://www.tut.edu.vn	url src	bad host

Obrázek 21 – Přehled nalezených neplatných odkazů nástrojem Broken Link

7 PŘÍPADOVÉ STUDIE

Případové studie byly zpracovány na základě vlastních zkušeností, interních dokumentů a komunikace se zástupci vybraných firem, zabývajících se vývojem softwarových aplikací.

7.1 FNZ s.r.o.

Společnost FNZ, sídlem nyní v britském Edinburghu, byla založena v roce 2002 na Novém Zélandě. FNZ vyvíjí a poskytuje technologie pro investiční platformy ve finančním sektoru. Své služby poskytuje bankám, pojišťovnám a dalším správcům majetku nyní již na třech kontinentech.

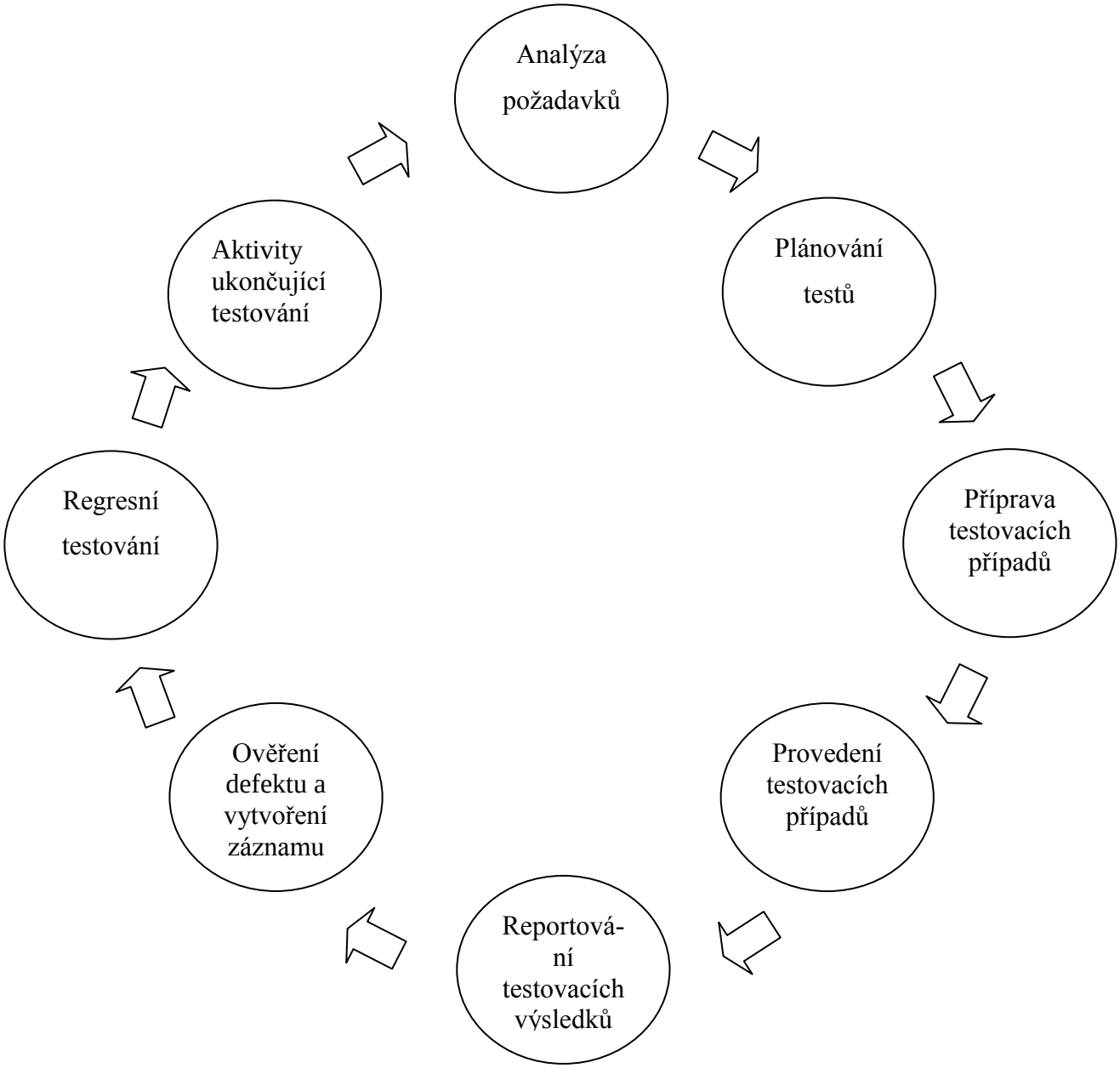
7.1.1 Popis testování

Společnost FNZ přizpůsobuje svůj produkt na míru každému zákazníkovi a jeho požadavkům. Pro práci na projektu každého zákazníka je vyčleněn tým, který má na starost testování aplikace. Přidávání nových funkcionalit a opravy zjištěných defektů probíhá pravidelným vydáváním kódu, release.

Každé vydání do produkce se skládá z několika fází, kde kód postupuje několika úrovněmi testovacích prostředí. V prvních úrovních vývojářský tým provádí testování jednotek a testování integrace komponent. V další úrovni specializovaný tým testerů provádí systémové testování a testování systémové integrace. Do další fáze testování jsou zapojeni zástupci klienta. Testovací prostředí v další úrovni je nejvíce podobné produkčnímu prostředí. Zde se provádí regresní testování a testování opravených klientských defektů.

FNZ využívá jak manuální tak automatické testování. Manuální testování je prováděno hlavně při ověřování nových funkcionalit, požadavků a oprav defektů. Automatické testování se využívá pro testování regrese a produkčního ověřování. Automatické regresní testy jsou psány na základě scénářů napsaných při manuálním testování a snaha je testování co nejvíce zautomatizovat.

7.1.2 Metoda testování



Obrázek 22 – Ilustrace testování vlastností, funkcí a systémových komponent ve společnosti FNZ

Analýza požadavků: Analýza požadavků obsahuje zhodnocení všech relevantních požadavků, které napomáhá při přípravě testovacích scénářů a určuje, zda jsou všechny stanovené požadavky jasné, úplné, konzistentní a jednoznačné. Všechny připravené scénáře by měly být mapovány k alespoň jednomu požadavku.

- Výstup: Testovací scénáře

Plánování testů: Plánování obsahuje všechny úkony spojené s přípravou publikací testovacích strategií, odhadů a detailních testovacích plánů. Tato činnost velmi napomáhá k dodržování projektového rozvrhu a rozvržení testovacích prostředků ke splnění plánu.

- Výstup: Detailní testovací plán, testovací odhady

Příprava testovacích případů: Návrh a příprava testovacích případů obsahuje všechny úkony spojené s analýzou řešení požadavků pro účely návrhu testovacích sad a testovacích případů v nich obsažených. Testovací případy reprezentují všechny požadované testovací varianty k pokrytí cílů testování. Po napsání testovacích případů jsou přezkoumány, aby byla zajištěna jejich správnost, úplnost a splnění FNZ standardů.

- Výstup: Testovací případy

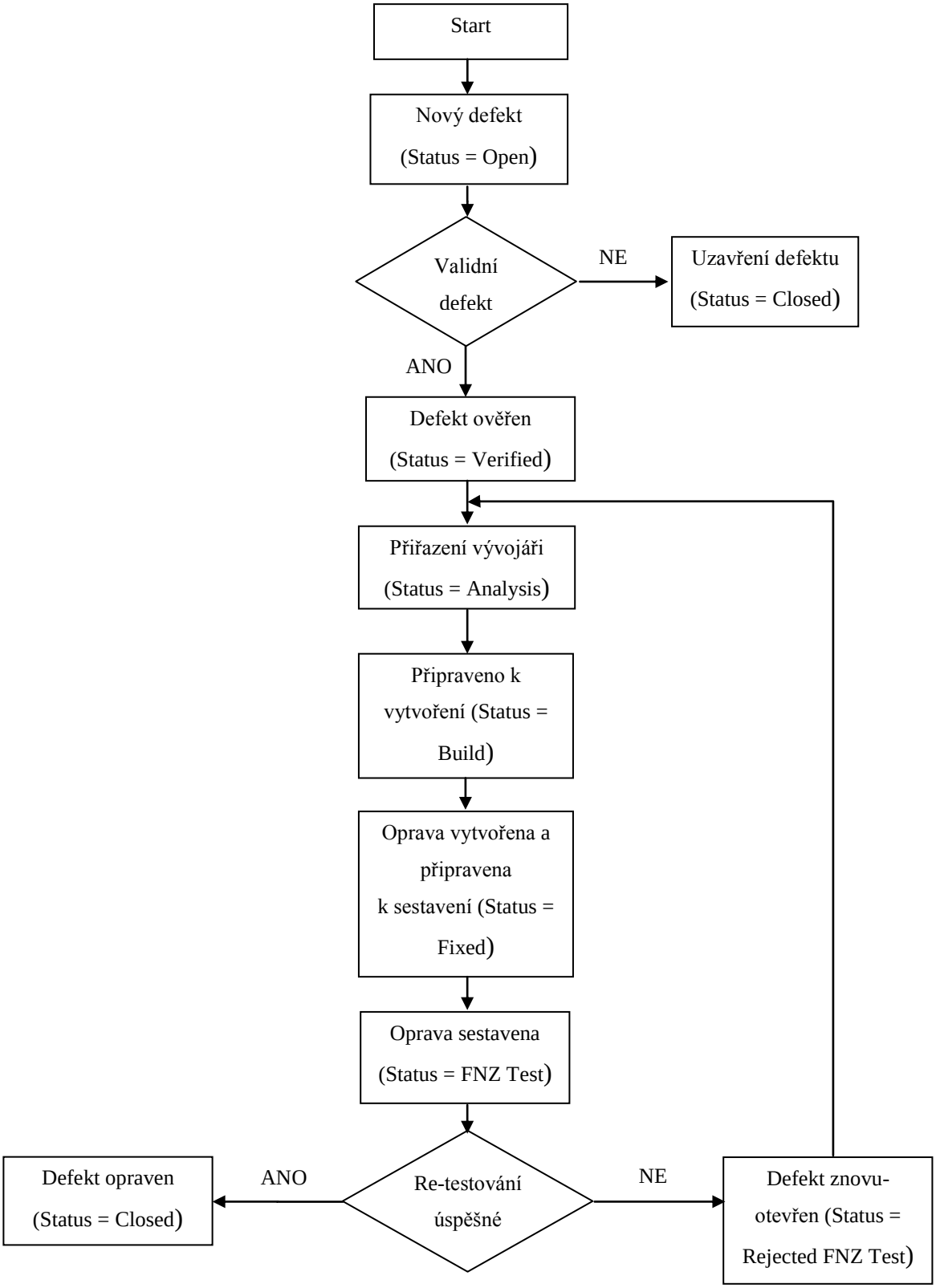
Provedení testovacích případů: Provedení testovacích případů zahrnuje veškeré úkony spojené s provedením nezbytných kroků v každém příslušném testovacím případu, za účelem ověření, že chování systému odpovídá požadavkům. Provádění testovacích případů generuje výsledky, odpovídající každému testu, stejně jako defekty. Defekty jsou spravovány testovacím nástrojem JIRA. Průběh testování je sledován a reportován.

- Výstup: Výsledky testovacích případů, Defekty, Záznam testovacích případů, Denní aktualizace stavu

Aktivity ukončující testování: V této závěrečné fázi se provádí veškeré úkony spojené s uzavřením testovacího úsilí. Vytvoří a zveřejní se závěrečná testovací zpráva. Tato zpráva obsahuje detaily týkající se testovaného pokrytí a jeho výsledků. Také popisuje profil nalezených a nevyřešených defektů na závěr testování. Dokončují se také všechny zbývající úkoly,

- Výstup: Závěrečná testovací zpráva

7.1.3 Životní cyklus defektu v FNZ



Obrázek 23 – Schéma životního cyklu defektu v testovacím procesu FNZ

Tabulka 3 – Přehled využívaných stavů defektu

Status:	Popis:
Open	Základní status nově vytvořeného defektu, když je očekáváno přiřazení.
Assigned	Defekt je přiřazen (většinou vývojáři) a očekává zpracování.
Info Required	Další informace o defektu jsou vyžadovány.
Verified	Defekt byl ověřen analytikem.
Fixed	Defekt byl opraven a očekává release.
FNZ Test	Defekt byl opraven, vydán a čeká na přetestování.
Rejected FNZ Test	Defekt byl přetestován a oprava chyby nebyla potvrzena.
Closed	Defekt byl uzavřen.
On Hold	Proces zpracování defektu byl pozdržen.

7.2 AVG Technologies

Společnost AVG, sídlem v nizozemském Amsterdamu, byla založena v roce 1991. AVG Technologies vyvíjí a vydává bezplatný i komerční bezpečnostní software pro koncové uživatele i firmy. Tato případová studie se zabývá jejím přístupem k testování vlastních webových stránek, jejímž nejdůležitějším bodem je elektronický obchod.

7.2.1 Popis testování

Společnost AVG využívá k testování 4 různá testovací prostředí, do kterých postupně integruje změny v software. V těchto prostředí se testuje jak za pomocí manuálního, tak i automatického testování. Testování provádí několik týmů, které můžeme podle oblasti testování rozdělit do 4 skupin. Jednotlivé skupiny mají na starost vždy jinou oblast aplikace.

7.2.1.1 Regular release

Změny a rozšíření aplikace je prováděno v pravidelných 14 denních intervalech ukončených rozhodnutím, zda je daná verze vhodná k vydání do produkce. V tomto případě se jedná o normální proces, tzv. Regular Release. V případě potřeby se využívají další dva typy release:

7.2.1.2 Operativní release

Mimo regulární release, který probíhá pravidelně, se pro případy drobných změn využívá operativních releasů. Tyto releasy nejsou pravidelné, ale předběžně jsou naplánovány 1-2x týdně. Při tomto releasu se přeskakuje integrační prostředí a prostředí simulující produkci a provedené změny jdou přímo z development prostředí do produkce.

7.2.1.3 Emergency release

Slouží pro případy potřeby akutních změn či oprav, které nelze odložit. Emergency releasy nejsou předem plánované, ale v případě potřeby je změny do produkce možné provést v rámci několika hodin.

7.2.1.4 Postup testování

Fáze testování v každé release začíná definicí požadavků a jejich rozdělením na příslušné týmy. Úkolem testera je na základě těchto požadavků vytvořit testovací případ v programu JIRA, podle kterého se pak testuje a hodnotí se provedení všech kroků. Pokud všechny kroky byly splněny, test se považuje za úspěšný. Pokud alespoň jeden krok splněn nebyl, celý test se považuje za neúspěšný a musí se vykonat úkony k nápravě.

7.2.2 Rozdělení týmů

Tým A: Tento tým má na starost ověřování funkce frontendu, jako elektronický košík, zda nákupy lze úspěšně dokončit. Testuje také finanční stránku obchodu, jestli jsou správně aktualizované ceny, jestli se generují faktury, atd. Testování probíhá manuálně, případně za pomocí nástrojů iMacro a Selenium.

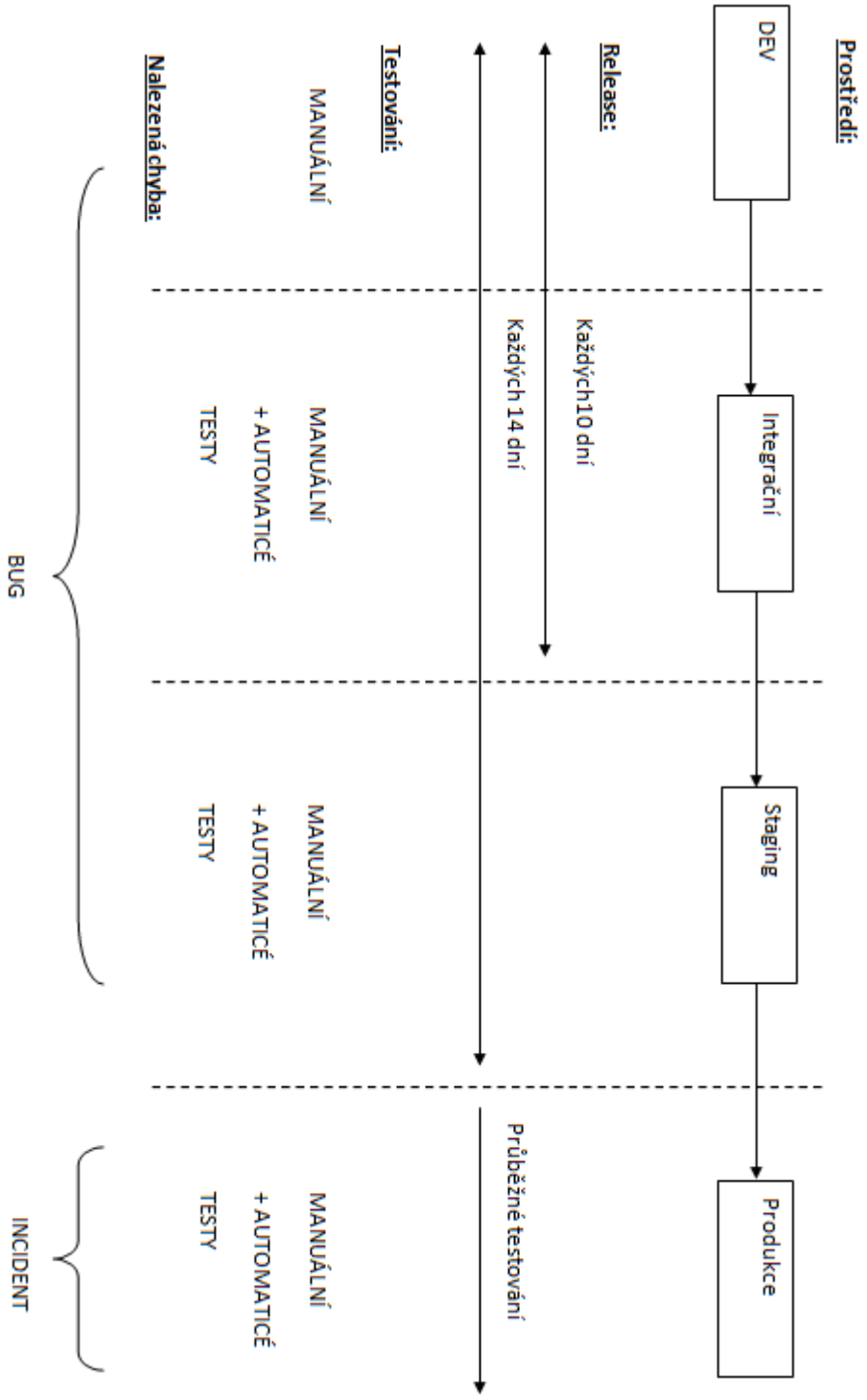
Tým B: Tento tým má na starost testování backendu elektronického obchodu, na kterou nemá běžný uživatel přístup, databázi s informacemi o uživatelích a licenčních čísel. Testuje se také připojení databáze a ověřování objednávek. Testování probíhá manuálně, případně za pomocí nástrojů interně vyvinutých.

Tým C: Tento tým má na starost testování oblasti uživatelských účtů. Testuje se, zda uživatelské účty mohou správně spravovat své objednávky a zakoupené produkty. Velká pozornost je také věnována testování správy licencí, jejich prodloužení, nebo upgradování. Testování probíhá manuálně, případně za pomoci nástroje SoapUI.

Tým D: Tento tým má na starost automatizaci testů, které provádějí všechny ostatní týmy. Na testování automatických testů se používá převážně Javascript. Pro správu testů se používá nástroj Bamboo, od stejné firmy jako JIRA. Tento tým kontroluje spouštění testů a vyhodnocuje jejich výsledky.

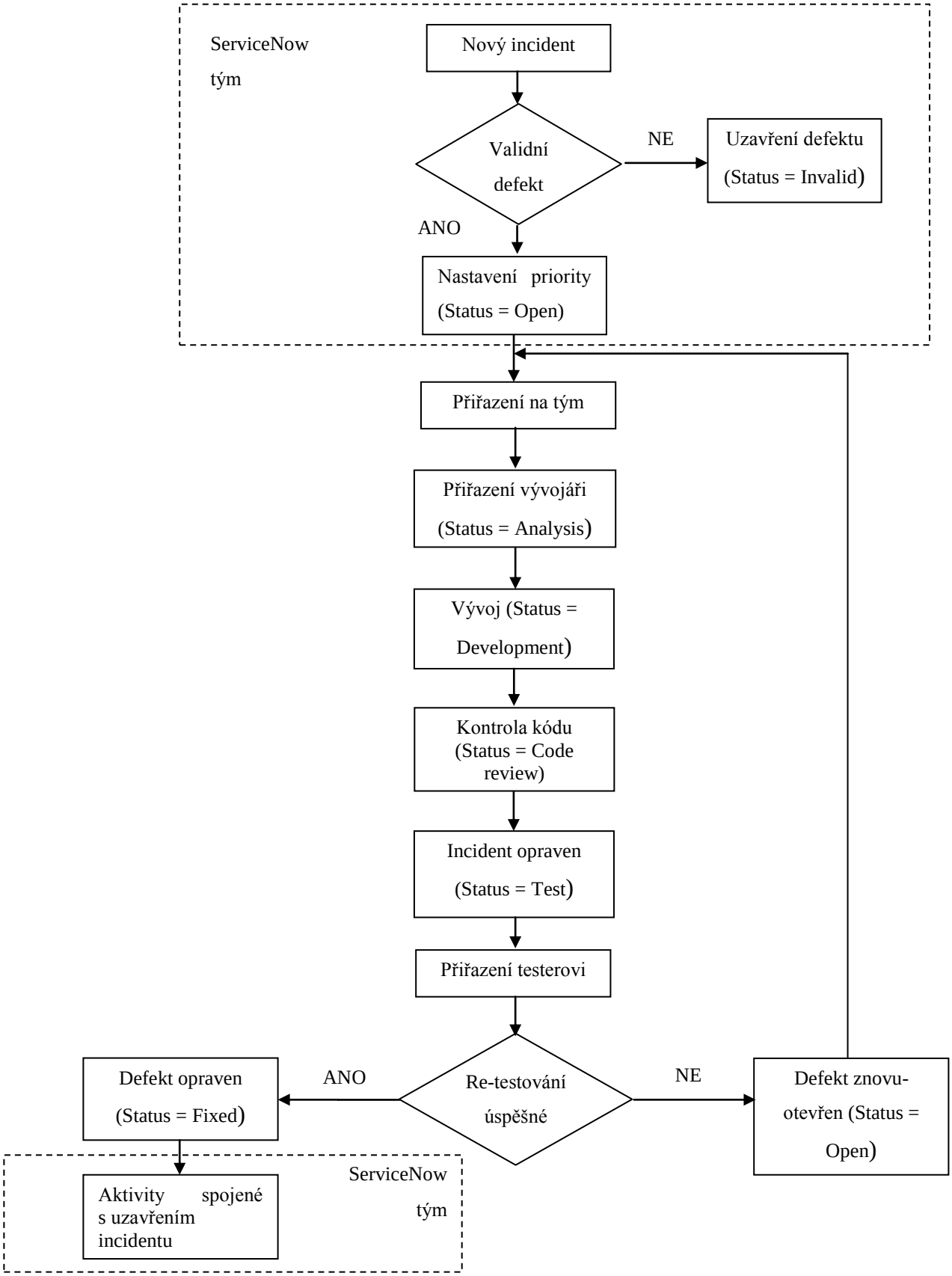
Pokud je nalezena chyba v aplikaci během testování ve vývojovém prostředí, je tato chyba označena jako bug a spravuje se interně za pomoci testovacího nástroje JIRA. Chyba nalezená v produkci je označena jako incident a má přidělenu větší prioritu. Správa incidentů probíhá přes nástroj ServiceNow, poté se incident vytvoří v JIRE a přiřadí příslušnému týmu.

7.2.3 Schéma



Obrázek 24 – Schéma procesu testování ve společnosti AVG

7.2.4 Životní cyklus defektu v AVG



Obrázek 25 - Schéma životního cyklu defektu v testovacím procesu AVG

Po nalezení incidentu, je pomocí nástroje ServiceNow založen incident. ServiceNow tým dále incident ověří, nebo uzavře jako invalidní. V případě, že je incident ověřen jako validní, je mu nastavena priorita a v JIRE vytvořen odpovídající issue. JIRA issue je přiřazeno konkrétnímu týmu, který zodpovídá za jeho vyřešení. Po opravení je změněný kód poskytnut jinému vývojáři, který provede kontrolu kódu. Poté následuje otestování a je rozhodnuto, zda byl incident úspěšně opraven. Pokud ano issue se předává ServiceNow týmu, který sepíše zprávu o nalezených příčinách, kde vznikl (frontend, backend) a jakým způsobem byl opraven. Pokud incident opraven nebyl, vrací se zpátky odpovědnému vývojáři.

8 NÁVRH METODY TESTOVÁNÍ

Na základě informací uvedených v teoretické části práce a zavedených procesů používaných v praxi, prezentovaných v případových studiích, je navržena metoda testování.

8.1 Popis testování

Testování by mělo začít co nejdříve, nejlépe hned u návrhu požadavků. Brzké testování může odstranit nesrovnalosti v návrhu a snížit tak náklady na jejich pozdější odstranění. Na základě otestovaných požadavků vývojář napíše kód, který přináší danou funkcionalitu. Před jeho vydáním a aktualizací aplikace, by měl být kód zkontrolován jiným vývojářem. Tato kontrola může pomoci s udržováním kódu splňující standardy a také odhalit chyby, které na první pohled nemusí být zřejmé. Po tomto ověření, bude kód vydán do testovacího prostředí, kde danou změnu, funkcionalitu nebo vlastnost aplikace bude testovat specializovaný tým.

Release do další úrovně testovacího prostředí by měl být pravidelný, s možností vyžádání mimořádného vydání kódu. Testovací prostředí jsou víceúrovňové. Na první úrovni testování provádí vývojáři integrační a jednotkové testy. V další úrovni je testování prováděno specializovanými týmy. Pokud aplikace splňuje požadavky po provedení testování v této úrovni, je kód vydán do další úrovně. V této úrovni je prostředí co nejvíce podobné tomu, které běží v produkci, a do testování v této úrovni je zapojen i klient. Zde se testuje chování aplikace v podmínkách simulujících reálné, provádí se přetestování produkčních defektů a regresní testování. V nejvyšší úrovni prostředí, v produkci, se na běžící aplikaci provádí automatické testování, ověřující korektnost chování aplikace v ostrém provozu.

Specializované testovací týmy mají vždy na starost určitou oblast aplikace, tak, aby znalosti o dané sekci byly co nejpřesnější. Pokud společnost vyvíjí produkt na stejném základě pro více zákazníků a uzpůsobuje ho jim na míru, budou mít tyto specializované týmy vždy na starost stejnou oblast aplikace napříč všemi klienty. Tím je také zaručeno efektivnější testování. Týmy budou rozděleny do 4 skupin.

Skupina A

- Týmy v této skupině provádí funkční testování a testování použitelnosti. Zabývat se tak budou například:
    - Ověření vstupu a výstupu dat
    - Verifikace textových polí (mezní hodnoty, speciální znaky, atd.)
    - Ověření grafického rozhraní.
    - Black box testování.

Skupina B:

- Týmy v této skupině provádí testování spojení s databází, konzistence dat a backend procesů. Zabývat se tak budou například:
    - Procesy běžícími na pozadí aplikace.
    - Administrativní správa a část aplikace, do které nemá běžný uživatel přístup.
    - Testování připojení k databázi a provádění SQL příkazů.

Skupina C:

- Týmy v této skupině provádí testování kompatibility, výkonu a bezpečnosti. Zabývat se tak budou například:
    - Zabezpečení přihlášení, certifikátů a webového serveru.
    - Testování proti známým slabinám v bezpečnosti jako SQL injection.
    - Zajišťuje dodržení zásad důvěrnosti, integrity a dostupnosti.

Skupina D:

- Týmy v této skupině provádí automatizaci testování a ověřování chodu aplikace v produkci. Zabývat se tak budou například:
    - Kontrola správného chodu aplikace v různých zeměpisných oblastech.
    - Pravidelně kontrolovat funkčnost a stav aplikace.

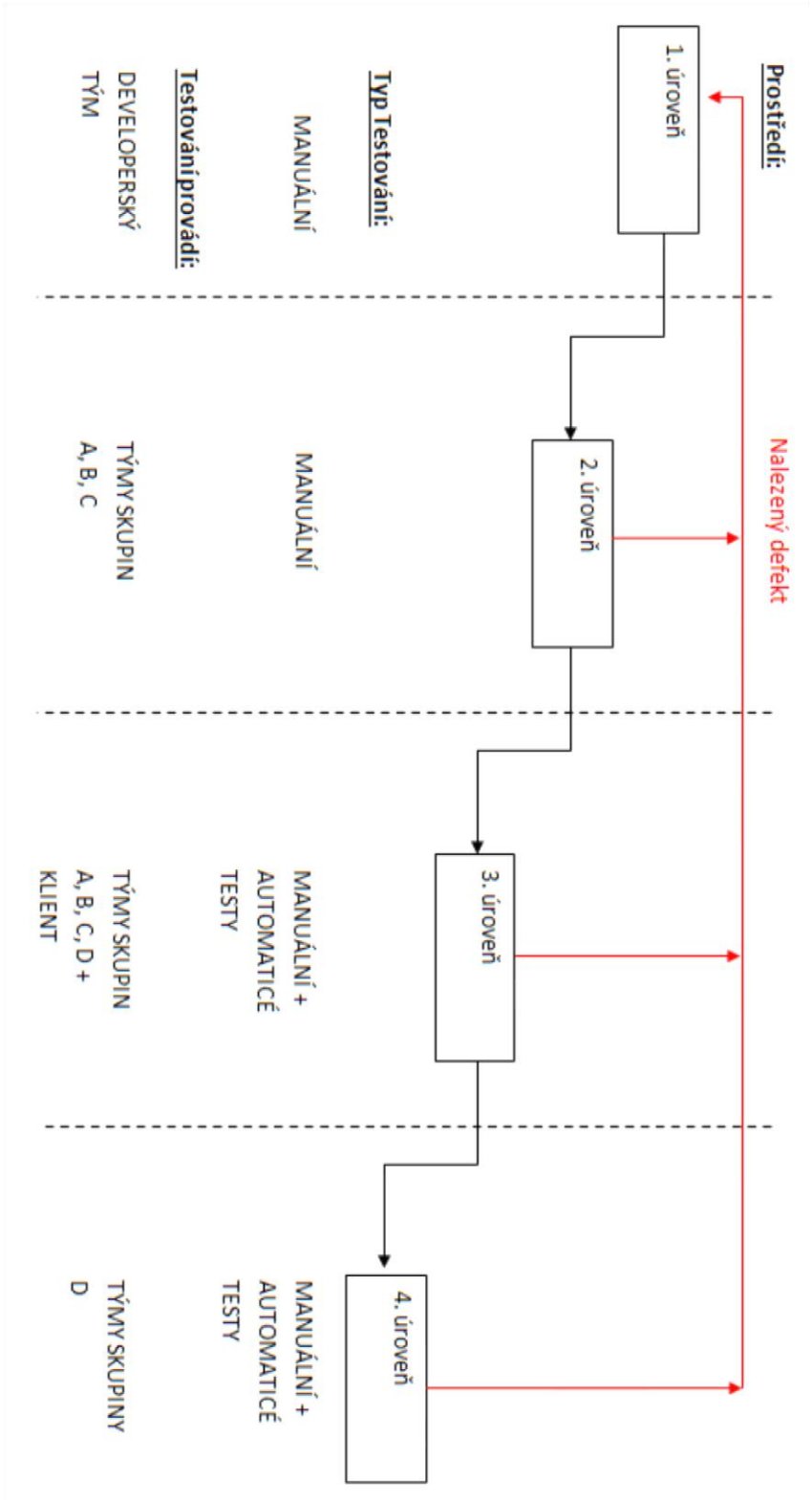
S rostoucí složitostí systému rostou také nároky na znalosti testovacích postupů a procesů každého testera. Testovací postupy se proto budou dokumentovat a budou dostupné v případě potřeby. Nalezeným chybám bude vždy přidělena priorita. Defekty s nejvyšší prioritou bude vývojář opravovat jako první, aby bylo zajištěno, že ty nejdůležitější budou vždy opraveny před následující release.

Tabulka 4 – Označení závažnosti nalezené chyby

Priorita:	Popis:
1 - Kritická	Kritická chyba způsobující pády systému nebo zabraňující dalšímu testování.
2 – Vážná	Vážná chyba způsobující nesplnění důležitého požadavku na aplikaci.
3 - Střední	Chyba, která nemá vážný dopad na chod aplikace.
4 - Mírná	Mírná chyba, například překlepy, chyby zarovnání atd.

Pro zlepšení efektivity testování budou použity výše popsané testovací nástroje, pomocí níž bude proces testování zaznamenáván a dokumentován.

8.2 Schéma



Obrázek 26 – Návrh procesu testování

ZÁVĚR

V teoretické části této práce jsem se pokusil srozumitelně a jednoduše nastínit základní pravidla a principy testování webových aplikací i softwarových aplikací obecně, jaké jsou jejich výhody, nevýhody, druhy a na čem jsou založeny tak, aby to bylo srozumitelné i čtenářovi, který se zatím s testováním softwarových aplikací nesetkal. Testování softwarových aplikací je velmi důležitá úloha a tato práce může posloužit jako odrazový bod pro čtenáře zamýšlející o věnování se této činnosti. Byly také popsány případy, kdy nedostatečné otestování programu vedlo k selhání produktu a následným finančním i materiálním ztrátám.

V praktické části této práce jsem se nejprve věnoval nástrojům používaných v testovacím procesu a některé z nich představil. Pomocí daných nástrojů byly provedeny dva testy na stránkách University Tomáše Bati. Nástrojem na hledání neplatných odkazů, Broken Link Checker, bylo objeveno celkem 18 upozornění na neplatné linky. Nástrojem na hledání zranitelných míst v zabezpečení webové aplikace, OWASP Zed Attack Proxy, bylo nalezeno celkem 7 upozornění, nicméně žádné z nich nebylo vyhodnoceno jako vážné. Stránky university byly také využity pro názornou ukázku využití dalšího nástroje sloužícímu k vytváření a provádění automatických testů. Byl vytvořen a proveden jednoduchý automatický skript na přihlášení a odhlášení uživatele.

Dále byly provedeny dvě případové studie, jak testují svůj produkt vybrané softwarové společnosti – FNZ s.r.o. a AVG Technologies. Součástí případových studií je i znázornění životního cyklu nalezené chyby. Tyto studie byly provedeny na základě vlastních zkušeností, dostupných interních materiálů a komunikace se zástupci firem.

S využitím informací zjištěných o testovacích procesích a metodách v daných firmách, byla navrhnuta vlastní metoda testování, která kombinuje to nejlepší z obou případových studií. Konkrétně se jedná o zapojení klienta do procesu testování a rozdělení specializovaných týmů podle oblasti testované aplikace.

CONCLUSION

In the theoretical part of this thesis I have tried to clearly and simply outline the basic principles of testing of web applications and software applications in general. Namely what are their advantages and disadvantages, types and on what are they based on. The purpose was to explain that in an understandable way so that even readers who are not familiar with software testing would be able to comprehend. Testing of software applications is a very important task and this work can serve as a starting point for readers intending to pursue this activity. There have also been described cases where insufficient testing of a program led to a failure of the product and subsequent financial and material losses.

In the practical part of this thesis I started with presenting the tools used in software testing. There were two tests performed on the Tomas Bata University website using those tools. The first tool, Broken Link Checker, was used to check for broken links on the university website. It has discovered a total of 18 warning of broken links. The second tool, OWASP Zed Attack Proxy, was used to search for security vulnerabilities on the university website. It has found a total of 7 warnings, however none of them was considered to be serious. The university pages were also used for demonstration of the use of another tool that allows to create and run automated tests scripts. A simple automatic script to logon and logoff user was created.

Two case studies were also conducted on how selected software developing companies, FNZ ltd and AVG Technologies, tests its product. Part of the case studies is a representation of the error life cycle. These studies were carried out on the basis of my own experience, the available internal documents and communication with company representatives.

Using the information gained about the testing process and methods in these companies an own testing method was designed. This testing method combines the best from both case studies. Specifically, the involvement of the client in the process of testing and using specialized functional teams.

SEZNAM POUŽITÉ LITERATURY

[1] Base 36 Smart Solutions. Automated vs. Manual Testing: The Pros and Cons of Each. [online]. [cit. 2014-05-20]. Dostupné z: http://www.base36.com/2013/03/automated-vs-manual-testing-the-pros-and-cons-of-each/

[2] BECK, Kent. Test-driven development: by example. Boston: Addison-Wesley, c2003, xix, 220 p. ISBN 03-211-4653-0.

[3] (EDITOR), Brian Hambling a Peter Morgan ... [et]. AL]. Software testing: an ISEB foundation. 2nd ed. London: British Computer Society, 2010. ISBN 978-190-6124-762.

[4] ELLIOTT, Geoffrey. Global business information technology: an integrated systems approach. New York: Pearson Addison Wesley, 2004, xvi, 503 p. ISBN 03-212-7012-6.

[5] EXPERIMENTUS, Intelligent Test Method (iTM) Demonstration Version: Fundamental Test Process. [online]. [cit. 2014-05-20]. Dostupné z: http://www.experimentus.com/itm/06_00120_Test_Planning_and_Control.htm

[6] GHAHRAI, Amir. Testing Excellence: Defect Clustering in Software Testing. [online]. [cit. 2014-05-25]. Dostupné z: http://www.testingexcellence.com/defect-clustering-in-software-testing/

[7] GHAHRAI, Amir. Testing Excellence: Seven Principles of Software Testing. [online]. [cit. 2014-05-25]. Dostupné z: http://www.testingexcellence.com/seven-principles-of-software-testing/

[8] GHOSHAL, Abhimanyu. A comprehensive guide to testing your Web app: How to get the most out of your sessions. [online]. [cit. 2014-05-20]. Dostupné z: http://thenextweb.com/apps/2013/11/28/guide-testing-web-app-steps-approach-testing-get-sessions/

[9] HETZEL, William C. The complete guide to software testing. 2nd ed. Wellesley, Mass.: QED Information Sciences, 1988, ix, 280 p. ISBN 08-943-5242-3.

[10] HOWEL, Rick. Software QA and Testing Frequently-Asked-Questions Part 2. [online]. [cit. 2014-05-20]. Dostupné z: http://www.softwareqatest.com/qatfaq2.html

[11] ISTQB Exam Certification: What is fundamental test process in software testing? [online]. [cit. 2014-05-20]. Dostupné z: http://istqbexamcertification.com/what-is-fundamental-test-process-in-software-testing

[12] ISTQB Exam Certification: What is V-model- advantages, disadvantages and when to use it? [online]. [cit. 2014-05-20]. Dostupné z: http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/

[13] JENKINS, Nick. A Project Management Primer: Basic Principles - Scope Triangle. [online]. [cit. 2014-05-20]. Dostupné z: http://www.projectsmart.co.uk/project-management-scope-triangle.php

[14] JOVANOVIĆ, Irena. Software Testing Methods and Techniques. [online]. [cit. 2014-05-20]. Dostupné z: http://www.internetjournals.net/journals/tir/2009/January/Paper%2006.pdf

[15] KANER, J.D., PH.D., Cem. Exploratory Testing. *Florida Institute Of Technology*, [online]. 2006 [cit. 2014-04-25]. Dostupné z: http://www.kaner.com/pdfs/ETatQAI.pdf

[16] MISHRA, Dwarika Dhish. Manual Testing Interview Question: Difference Between Error, Defect, and Failure. [online]. [cit. 2014-05-20]. Dostupné z: http://www.abodeqa.com/2012/11/05/manual-testing-interview-question-difference-between-error-defect-and-failure/

[17] MYERS, Glenford J. The art of software testing. New York: Wiley, c1979, xi, 177 p. ISBN 04-710-4328-1.

[18] NIELSEN, J. Usability engineering. Vyd. 1. Boston: AP Professional, 1993, 362 s. ISBN 01-251-8406-9.

[19] OLADIMEJI, Patrick. Levels of testing: Advance topics in Computer Science. *University of Wales Swansea Computer Science Department* [online]. [cit. 2014-05-25]. Dostupné z: http://www.cs.swan.ac.uk/~csmarkus/CS339/presentations/20061202_Oladimeji_Levels_of_Testing.pdf

[20] ORLOFF, Jeff. IBM: Developer Works: Web application security. Testing for vulnerabilities. [online]. [cit. 2014-05-20]. Dostupné z: http://www.ibm.com/developerworks/library/wa-appsecurity/

[21] PALANI, Gowri Shankar. IBM: Developer Works. Summary of web application testing methodologies and tools. [online]. [cit. 2014-05-20]. Dostupné z: http://www.ibm.com/developerworks/library/wa-webapptesting/

[22] PATTON, Ron a Peter Morgan ... [et]. AL]. Software testing: an ISEB foundation. 2nd ed. Indianapolis, Ind.: Sams, c2001, xi, 389 p. ISBN 06-723-1983-7.

[23] PMI, A guide to the project management body of knowledge: (PMBOK guide). 4th ed. Newton Square: Project Management Institute, c2008, xxvi, 467 s. ISBN 978-1-933890-51-7.

[24] SHINDE, Vijay. Software Testing Help: Types of software Testing. [online]. [cit. 2014-05-20]. Dostupné z: http://www.softwaretestinghelp.com/types-of-software-testing/

[25] SHINDE, Vijay. Software Testing Help: Web Testing: Complete guide on testing web applications. [online]. [cit. 2014-05-20]. Dostupné z: http://www.softwaretestinghelp.com/web-application-testing/

[26] Software Process Models. *The University of West Indies at Cave Hill, Barbados* [online]. [cit. 2014-04-04]. Available at: http://www.cavehill.uwi.edu/staff/eportfolios/paulwalcott/courses/comp2145/2009/software_process_models.htm

[27] SOMMERVILLE, Ian. Software engineering: (PMBOK guide). 9th ed. Boston: Pearson, c2011, xv, 773 p. ISBN 978-013-7053-469.

[28] Source of Acquisition NASA Johnson Space Center: Error Cost Escalation Through the Project Life Cycle. [online]. [cit. 2014-05-25]. Dostupné z: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036670.pdf

[29] Target, The-Software-Experts: Software Process Models. [online]. [cit. 2014-05-20]. Dostupné z: http://www.the-software-experts.com/e_dta-sw-process.php

[30] Testing introduction: Software Engineering study materials. *North Carolina State University* [online]. [cit. 2014-04-25]. Dostupné z: http://agile.csc.ncsu.edu/SEMaterials/IntrotoTesting.pdf

[31] Tutorialspoint: Levels of Testing. [online]. [cit. 2014-05-20]. Dostupné z: http://www.tutorialspoint.com/software_testing/levels_of_testing.htm

[32] Tutorialspoint: SDLC V-Model. [online]. [cit. 2014-05-20]. Dostupné z: http://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

[33] Tutorialspoint: Software Testing Documentation. [online]. [cit. 2014-05-20]. Dostupné z: http://www.tutorialspoint.com/software_testing/testing_documentation.htm

[34] Tutorialspoint: Software Testing Methods. [online]. [cit. 2014-05-20]. Dostupné z: http://www.tutorialspoint.com/software_testing/testing_methods.htm

[35] Tutorialspoint: Software Testing Types. [online]. [cit. 2014-05-20]. Dostupné z: http://www.tutorialspoint.com/software_testing/testing_types.htm

SEZNAM OBRÁZKŮ

Obrázek 1 - Trojúhelník znázorňující vztah mezi projektovými náklady, kvalitou a časem [13] 14
Obrázek 2 – Základní testovací proces [3] 17
Obrázek 3 – Vodopádový model [3] 21
Obrázek 4 – Model iteračního vývoje [3] 22
Obrázek 5 – Model programování řízeného testy [2] 24
Obrázek 6 - Grafické znázornění V-modelu [3] 25
Obrázek 7 – Relativní náklady na odstranění chyby v závislosti na fázi objevení chyby [28] 28
Obrázek 8 – Schéma integrace typu Shora-dolů [3] 33
Obrázek 9 – Schéma integrace Zdola-nahoru [3] 34
Obrázek 10 – JIRA dashboard 45
Obrázek 11 – JIRA issue 46
Obrázek 12 – JIRA issue – spodní sekce 46
Obrázek 13 – Grafické rozhraní pluginu Selenium IDE 48
Obrázek 14 – Panel nástrojů Selenium IDE 48
Obrázek 15 – Parametry vytvořeného skriptu 49
Obrázek 16 – Výsledek spuštění automatického skriptu 50
Obrázek 17 – Výsledky skenování programu ZAP 51
Obrázek 18 – Upozornění na nedostatky v zabezpečení 51
Obrázek 19 – Zadání adresy do vyhledávače Broken Link 52
Obrázek 20 – Zadání bezpečnostního kódu na stránce nástroje Broken Link 53
Obrázek 21 – Přehled nalezených neplatných odkazů nástrojem Broken Link 54
Obrázek 22 – Ilustrace testování vlastností, funkcí a systémových komponent ve společnosti FNZ 56
Obrázek 23 – Schéma životního cyklu defektu v testovacím procesu FNZ 58
Obrázek 24 – Schéma procesu testování ve společnosti AVG 62
Obrázek 25 - Schéma životního cyklu defektu v testovacím procesu AVG 63
Obrázek 26 – Návrh procesu testování 68

SEZNAM TABULEK

Tabulka 1 – Srovnání výhod a nevýhod black box metody [34] 36
Tabulka 2 - Srovnání výhod a nevýhod white box metody [34] 37
Tabulka 3 – Přehled využívaných stavů defektu 59
Tabulka 4 – Označení závažnosti nalezené chyby 67