Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Testování software ve fázi vývoje

Bc. Jakub Lakomý

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jakub Lakomý

Informatika

Název práce

Testování software ve fázi vývoje

Název anglicky

Software testing in development process

Cíle práce

Hlavním cílem diplomové práce je na základě studia odborné a vědecké literatury navrhnout odpovídající testy pro konkrétní aplikaci a provést vyhodnocení výsledků testů. Dílčím cílem bude navrhnout vhodné metodiky vývoje testování software na jejichž základě bude proveden způsob zvolení postupu testování.

Metodika

V teoretické části práce bude proveden sběr dat a informací, jejich studium a následná interpretace získaných informací na základě analýzy a syntézy dané problematiky. Takto získané informace vytvoří předpoklady pro zpracování praktické části, ve které bude testována vybraná aplikace. Na základě výsledků provedených testů budou identifikované možné nedostatky aplikace a navržena vhodná řešení pro jejich odstranění.

Doporučený rozsah práce

60 stran

Klíčová slova

testování, metodika, software, analýza, defect, vývoj

Doporučené zdroje informací

KADLEC, Václav. Agilní programování: metodiky efektivního vývoje softwaru. 1. vydání. Brno: Computer Press, 2004. ISBN 80-251-0342-0

LEWIS, William E a Gunasekaran VEERAPILLAI. Software testing and continuous quality improvement. 2nd ed. Boca Raton: Auerbach Publications, 2005. ISBN 08-493-2524-22

PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. Jak testuje software Microsoft. Vyd. 1. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5

PATTON, Ron. Testování software. Praha: Compiter press, 2002. ISBN 80-7226-636-5

STEPHENS, Matt a Doug ROSENBERG. Testování softwaru řízené návrhem. Vyd. 1. Brno: Computer Press, 2011. ISBN 978-80-251-3607-2

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Edita Šilerová, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 13. 11. 2017

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 13. 11. 2017

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 28. 03. 2018

Čestné prohlášení

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

V Praze dne 29.3.2018 _____

Poděkování

Rád bych touto cestou poděkoval své vedoucí práce paní Ing. Editě Šilerové, Ph.D., z katedry informačních technologií ČZU v Praze, za vedení a konzultace, které mi poskytla, dále své rodině za vytvoření pohodlného prostředí a podporu při vytváření práce.

Testování software ve fázi vývoje

Abstrakt

Práce řeší problematiku testování softwaru ve fázi vývoje. Hlavní pozornost je v tomto komplexním procesu věnována procesu testování z pohledu vytvoření strategie testování, návrhu a exekuce konkrétních testů.

Teoretickou část tvoří popis životního cyklu vývoje softwaru, používané modely a využívané metodiky při jeho tvorbě. Tato část práce je orientována na návrh plánu testování, jsou zde charakterizovány techniky testování a jednotlivé typy testů podle úrovně vývoje. V další části jsou shrnuty poznatky z oblasti vytváření testovacích scénářů a zaznamenání chyb nalezených na základě jejich provádění.

Praktická část práce řeší testování konkrétní aplikace, na základě získaných informací z teoretické části je sestaven plán a je definovaná strategie testování. Dostupná dokumentace k testovaným funkcionalitám tvoří předpoklady pro vypracování specifikace případů užití na jejichž základě jsou vytvořeny konkrétní testovací scénáře.

Samotné testování je realizováno navrženými testy, paralelně s tímto testováním jsou určeny rizikové oblasti aplikace, které jsou podrobeny exploratornímu testování. Nalezené chyby jsou zaznamenány v aplikaci Trello nebo jsou reportovány analytickému týmu z důvodu zajištění jejich budoucí opravy.

V závěru práce jsou zhodnoceny dosažené výsledky a je rozhodnuto o celkovém výsledku projektu, jsou zde popsány nalezené chyby s doporučenými možnostmi pro jejich odstranění.

Klíčová slova: testování, metodika, software, analýza, defekt, vývoj

Software testing in development process

Abstract

Thesis is pursuing problems of software testing during the phase of development. Main focus, in this complex process, is on the procedure of testing from the design of testing strategy point of view, concept and execution of particular tests.

Theoretical part consists of description of life cycle of software development, models and methodologies used during the production. This part of the project is focused on creating of a testing plan. It contains the description of testing techniques and individual types of tests sorted by level of development. In the next section is the summary of findings from the area of creation of testing scripts and bug tracking found on the basis of their execution.

Practical part of the concept solves testing of the particular application. On the basis of gained information from the theoretical part the plan is built and strategy of testing is defined. Available documentation for the tested functionalities form prerequisites for designing specification of use cases on the basis of which there are made concrete testing scenarios.

Testing on itself is brought into effect by designed tests, parallel to this testing there are set high risk regions of the application, which are submitted to exploratory testing. Found bugs are noted in the Trello application or reported to analytical team for reasons of securing their future repairs.

In the conclusion of the thesis there is an evuluation of the achieved results and the whole outcome of the project is reviewed. It gives a list of discovered bugs together with recommended solutions.

Keywords: testing, methodology, software, analysis, defect, development

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Software je v dnešním světě informačních technologií neopomenutelným prvkem, setkáváme se s ním prakticky v každé oblasti lidské činnosti od různých bankovních aplikací, softwaru ve zdravotnictví, až třeba po software v dopravním průmyslu.

Pokud software nepracuje, tak jak se od něj očekává nebo obsahuje chyby, může dojít ke značným nepříjemnostem, v oblasti bankovnictví například k finančním ztrátám, v dopravním průmyslu, hlavně nyní s rozvojem autonomních vozidel, může chyba způsobit i smrt.

V diplomové práci je řešeno zajištění kvality softwaru ve fázi jeho vývoje za předpokladu, že je kladen důraz na jeho testování, jak již napovídá název práce. Před návrhem samotných testovacích případů a exekucí testů je vytvořen plán testování, který definuje zvolenou testovací strategii, techniky a typy prováděných testů.

Donedávna byl proces testování aktivitou, které se při vývoji software nevěnovala taková pozornost jako dnes. V současné době má většina firem vlastní testovací tým nebo využívá outsourcingu od firem, které tým specialistů dodají a za kvalitu vyvíjeného software zodpovídají. Úkolem testovacího týmu, pak bývá návrh testovací strategie, příprava testů, jejich řízení a vykonávání. Výsledky provedených testů jsou zaznamenány a nalezené chyby jsou případně reportovány za účelem zajištění nápravy.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce je na základě studia odborné a vědecké literatury navrhnout odpovídající testy pro konkrétní aplikaci a provést vyhodnocení výsledků testů.

Dílčím cílem bude navrhnout vhodné metodiky vývoje testování softwaru na jejichž základě bude zvolen způsob postupu testování.

2.2 Metodika

V teoretické části práce bude proveden sběr dat a informací, jejich studium a následná interpretace získaných informací na základě analýzy a syntézy dané problematiky. Takto získané informace vytvoří předpoklady pro zpracování praktické části, ve které bude testována vybraná aplikace. Na základě výsledků provedených testů budou identifikované možné nedostatky aplikace a navržena vhodná řešení pro jejich odstranění.

3 Teoretická východiska

V teoretické části práce budou představeny základní pojmy, modely a metodiky užívané v procesu vývoje. Další část se bude věnovat samotnému procesu testování softwaru.

3.1 Životní cyklus vývoje softwaru

Proces vývoje softwaru známý také pod pojmem životní cyklus software *(„Software development life cycle“*) je sled úkolů, které definují procesy v každém kroku vývoje softwaru. Životní cyklus jako proces obsahuje 6 po sobě jdoucích hlavních kroků: [1]

1. Sběr a analýza požadavků
2. Návrh
3. Implementace
4. Testování
5. Nasazení do provozu
6. Údržba

Sběr a analýza požadavků

Jedná se o fázi, ve které shromažďují a analyzují veškeré byznysové požadavky. Je zaměřena hlavně na členy projektového týmu a další účastníky zúčastněných stran. Snahou této fáze je najít odpovědi na obecné otázky tykající se vývoje software. Po shromáždění požadavků jsou tyto požadavky analyzovány a zkoumá se možnost jejich začlenění do systému, který má být vyvíjen. Na konci této fáze je vytvořen dokument specifikace požadavků, který pak slouží pro účely pokynů pro další fáze projektu. [1]

Návrh

V této fázi se připraví návrh systému z požadavků, které byly shromážděny v předchozí fázi. Návrh pomáhá při určení hardwarových a systémových požadavků a při definování celkové architektury systému. Tato specifikace potom slouží jako vstup do další fáze.

Na konci této fáze se začíná připravovat testovací strategie ve které se plánuje především co a jak se bude testovat. [1]

Implementace

Jedná se o nejdelší fázi celého životního cyklu vývoje. V této fázi vystupují hlavně vývojáři, jedná se tedy o část procesu, ve které je samotný systém rozdělen do jednotlivých modulů a vyvíjen – programován. [1]

Testování

Po naprogramování jednotlivých modulů nebo systémů je na základě definice požadavků systém testován. V této fázi se ověřuje, jestli systém splňuje předem definované požadavky shromážděné během fáze sběru požadavků. Přicházejí zde na řadu všechny typy funkčních, ale i nefunkčních testů. [1]

Jednotlivým typům testů se bude podrobněji věnovat pozdější kapitola.

Nasazení do provozu

Po úspěšném otestování je systém předán zákazníkovi. Po předání zákazníkovi je systém podroben testům, ale již na straně zákazníka. Pokud jsou požadovány nějaké změny nebo se naleznou nesrovnalosti se specifikací, jsou tyto požadavky na změny předány zpět vývojovému týmu k přepracování. Jakmile jsou tyto změny implementovány dochází ke konečnému nasazení systému do provozu. [1]

Údržba

Nasazením do provozu fáze životního cyklu nekončí, systém je potřeba udržovat, čas o času se mohou objevit skutečné problémy, které nebyly zřejmé z předchozích fází. Tyto nedostatky je potřeba opravit a udržovat tím tak celý systém. [1]

3.2 Modely životního cyklu vývoje softwaru

Jak již bylo řečeno v předchozí kapitole testování je jedním z klíčových kroků v životním cyklu vývoje. Pro zvýšení efektivnosti testování není potřeba znát konkrétní metodiky vývoje, ale spíš aplikované modely životního cyklu vývoje software.

Z pohledu velikosti projektů od menších projektů až po komplexní korporátní aplikace vznikly časem různé modely a přístupy. Modely specifikují jednak kooperaci všech členů vývoje systému, ale definují i jednotlivé procesy a jejich vzájemnou návaznost.

Každý model má své výhody i nevýhody je proto důležité zvolit správný přístup podle toho jaký systém je vyvíjen. Podle vybraného modelu jsou v konkrétní fázi testování zvoleny různé typy testů. [2]

Ron Patton [2] ve své knize uvádí 4 hlavní modely:

1. Modle velkého třesku
2. Model programuj a opravuj
3. Vodopádový model
4. Spirálový model

Další modely, jak Patton [2] uvádí, vychází z těchto hlavních. Z důvodů využití variant těchto modelů v praxi budou proto v této práci představeny i ony.

3.2.1 Model velkého třesku

Oproti ostatním modelům není tento model přesně strukturovaný, tento model neobsahuje plánování, analýzu ani testování. Jedná se o zjednodušený model, který je především zaměřený na vývoj. Výhoda tohoto modelu je zejména jeho aplikovatelnost na systémy, kde nedokážeme přesně specifikovat vstupní požadavky a neznáme ani jeho přesný termín dokončení vývoje. [3]

3.2.2 Model programuj a opravuj

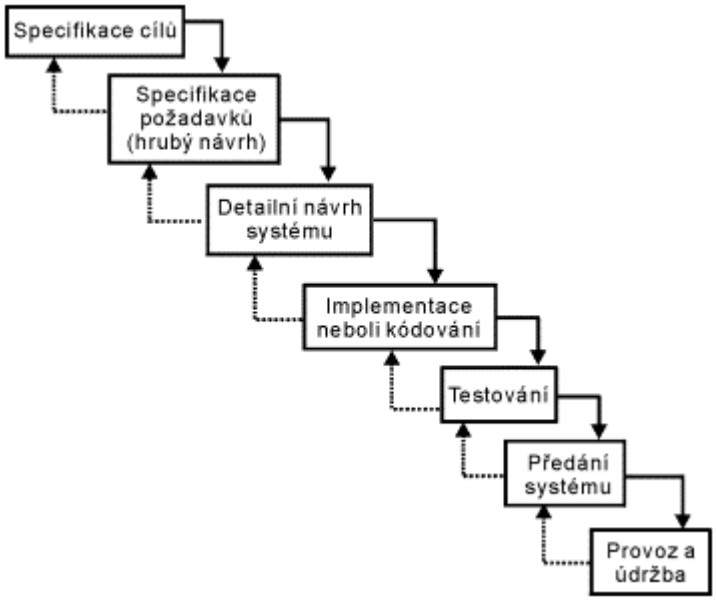
Opět se nejedná o velice efektivní model, je to však krok kupředu od předchozího modelu, jelikož musíme mít přinejmenším představu o tom, jaké budou vyžadovány požadavky na systém.

Tento přístup k vývoji software se především hodí pro malé systémy, které mají být rychle vytvořeny. Na začátku se pracuje jen s hrubou představou o tom, jak bude systém fungovat, vytvoří se nějaký jednoduchý návrh, po kterém následuje proces programování a opravování, který se opakuje. V určitém okamžiku se pak vývoj zastaví a vznikne tak dokončený systém, který je spíš prototypem nebo ukázkou než hotovým systémem. [2]

3.2.3 Vodopádový model

Vodopádový model je z předchozích modelů, prvním modelem, který je systematický, jedná se o tradiční lineárně – sekvenční model životního cyklu. Model tvoří několik po sobě jdoucích etap. V tomto modelu musí být každá etapa ukončena před přesunem do další fáze. Na konci každé fáze probíhá analýza, zda je možné přejít v procesu vývoje systému dál.

Testování v tomto modelu tvoří jednotlivou fázi, která nastává až po dokončení implementace systému. [4]



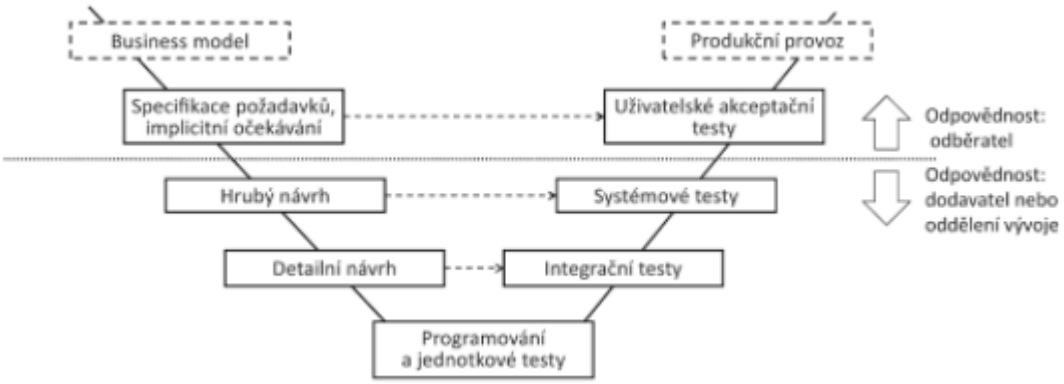
Obrázek 1 - Vodopádový model [5]

3.2.4 V model

Jedná se o model verifikace a validace. Stejně jako model vodopádu postupuje tento model sekvenčně. Každá fáze musí být ukončena před zahájením další fáze. Testování v tomto modelu je plánováno souběžně s odpovídající fází vývoje.

Plánování a vytváření testů probíhá v tomto modelu před fází samotného programování, čímž se ušetří čas a je zde větší šance na úspěch oproti vodopádovému modelu. Na levé straně modelu se nacházejí aktivity související s návrhem a specifikací, na pravé straně modelu se nachází aktivity spojené s testováním.

Výhodou tohoto modelu je právě to, že testování probíhá v jednotlivých etapách vývoje systému, zvyšuje se tím tedy šance na objevení chyb jak v implementaci, tak i v návrhu. [6]



Obrázek 2 - V model [6]

3.2.5 Inkrementální model

Inkrementální model nepracuje se systémem jako s celkem. Systém je rozdělen na několik modulů. Odehrává se tedy několik fází vývoje. Dalo by se říct, že se jedná o model vícenásobného vodopádového modelu.

Každý modul systému prochází fází sběru požadavků, návrhu, implementace a testování. Po nasazení prvního modulu vzniká první verze systému, ostatní moduly pak přidávají takovému systému další funkce. Postupně se tedy implementují další a další moduly, dokud není systém úplný. [7]

3.2.6 RAD model

„Rapid application development“ (RAD) model neboli model rychlého vývoje aplikací je typ inkrementálního modelu. Podobně jako jiné agilní modely reaguje na rigidnost klasického vodopádového modelu. Jeho inkrementální přístup snadněji reaguje na možné nové skutečnosti, které mohou být odhaleny až v průběhu projektu.

Tento přístup k vývoji zabraňuje selhání, kdy může být po několika měsících, či letech analýz odhalen, nějaký kritický problém až v pozdějších fázích. Při RAD modelu totiž vznikají průběžně prototypy, které si může koncový uživatel vyzkoušet. [8]

3.2.7 Spirálový model

Spirálový model je podobný inkrementálnímu přístupu k vývoji systému, klade však větší důraz na analýzu rizik. Spirálový model obsahuje čtyři hlavní fáze: analýza, hodnocení, vývoj a plánování. Vývoj systému prochází těmito fázemi v několika iteracích (spirálách). Vývoj systému probíhá od hrubé specifikace požadavků, v pozdějších fázích je tato specifikace i po konzultaci se zákazníkem více upřesňována. [9]

Fáze analýzy

V této fázi probíhá analýza rizik, provádí se zde identifikace rizik a alternativních scénářů. Na konci fáze se vytvoří prototyp, pokud se během této fáze objeví riziko, navrhnou a implementují se alternativní řešení. [9]

Fáze hodnocení

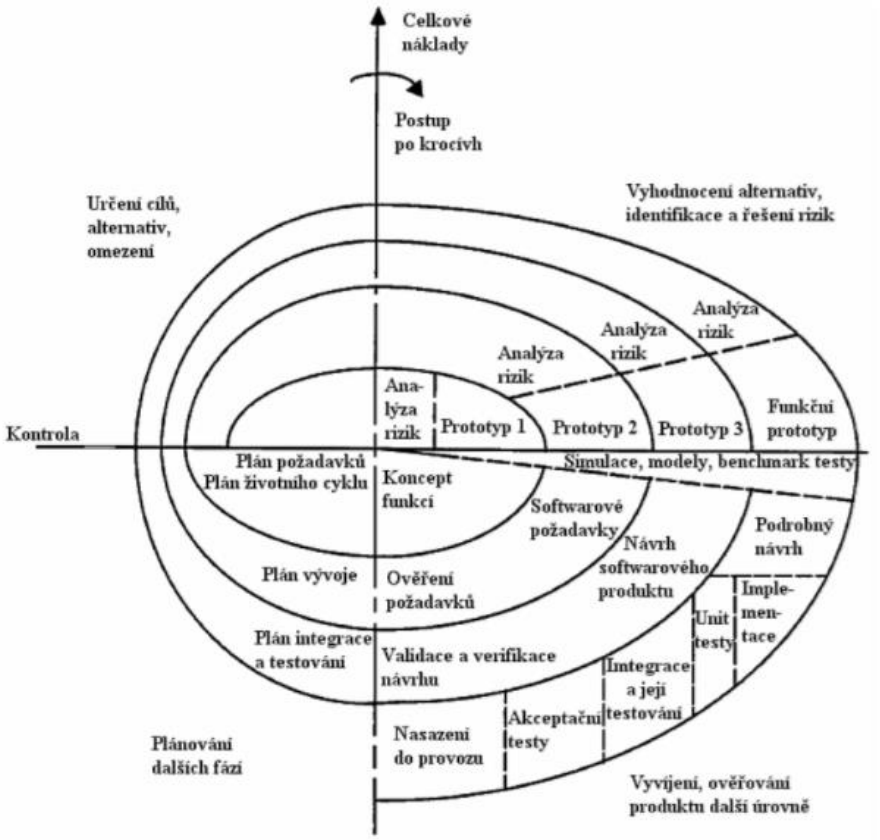
Tato fáze vývoje životního cyklu systému slouží zákazníkovi k vyhodnocení projektu, před tím, než se vstoupí do další iterace – spirály. [9]

Fáze vývoje

Zde dochází k samotnému programování aplikace, na konci této fáze, před vstupem do další dochází k testování. [9]

Fáze plánování

Tato fáze slouží ke sběru byznysových požadavků, po získání požadavků dochází ke specifikaci systémových požadavků na systém a přechází se do další fáze spirály. [9]



Obrázek 3 - Spirálový model [10]

3.3 Metodiky vývoje a testování software

Nejdříve bude proveden popis a výběr metodik pro testování. S ohledem na metodiky vývoje a testování software je metodika definována jako souhrn metod a postupů pro realizaci určitého úkolu. Takové metodiky pak bývají označovány jako metodiky vývoje.

Metodika budování IS/ICT definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu informačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení. [11]

Vývoj kvalitního softwaru závisí na řadě faktorů, jeden z nedůležitějších, který kvalitu vývoje software ovlivňuje, je proces testování.

V současné době jsou ve vývoji systémů sledovány dva hlavní proudy v metodických přístupech. Jedním proudem jsou metodiky rigorózní, druhým proudem, protikladem k rigorózním metodikám jsou agilní metodiky. [11]

Rigorózní metodiky

Rigorózní metodiky, vychází z přesvědčení, že procesy při vývoji lze řídit, měřit a plánovat. Tyto metodiky se snaží co nejlépe popsat a přesně definovat činnosti, procesy a vytvářené produkty, proto jsou často velice objemné. Tyto metodiky vycházejí z tradičního sekvenčního vodopádového modelu, ale existují i rigorózní metodiky založené na inkrementálním nebo iterativním vývoji. Zpětná vazba mezi fázemi vývoje je zajišťována prostřednictvím řízení změn. [11]

Agilní metodiky

Změny technologií a ekonomického prostředí, ke kterým dochází v tomto dynamickém světě vyžadují rychlé změny ve vývoji systému, tudíž i změny v metodikách. Tradiční rigorózní metodiky v tomto směru přestávají vyhovovat a začínají se prosazovat metodiky, které naopak od rigorózních metodik dokáží pružně reagovat na časté změny ve vývoji a přizpůsobovat jej měnícím se požadavkům. Takové metodiky se nazývají agilní. Existuje mnoho agilních metodik, jejich společnou myšlenkou však bývá rychlý vývoj systému nebo jeho části a prezentace zákazníkovi.

Na základě zpětné vazby pak dochází k úpravě procesů. Každá agilní metodika je svým způsobem specifická, ve finále ale všechny vycházejí ze stejných principů a hodnot. Tyto přístupy se jejich představitelé rozhodli roku 2001 podepsat a vytvořit tak „Manifest agilního vývoje software“. [11]

Tento manifest obsahuje 12 hlavních principů agilních metodik: [14]

- Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
- Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
- Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
- Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
- Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
- Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
- Hlavním měřítkem pokroku je fungující software.
- Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
- Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
- Jednoduchost – umění maximalizovat množství nevykonané práce je klíčové.
- Nejlepší architektury, požadavky a návrhy vzejdou ze samoorganizujících se týmů.
- Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

3.3.1 Rational Unified Process (RUP)

Metodika *„Rational Unified Process“* (RUP) patří mezi jednu z nejpoužívanějších rigorózních metodik, je založena na tzv. nejlepších praktikách vývoje softwaru: [12]

- iterativní vývoj
- řízení požadavků
- použití komponentové architektury
- vizuální modelování
- kontrola kvality software
- řízení změn

Iterativní vývoj

Vývoj by měl být rozdělen do iterací, přičemž by měla být dostupná po skončení každé iterace spustitelná verze aplikace.

Řízení požadavků

Cílem této praktiky je efektivní sběr požadavků, který probíhá mezi zadavatelem a odběratelem, umožňuje tak upřesňování zadání během vývoje projektu, což snižuje riziko neúspěchu projektu.

Použití komponentové architektury

Vývoj je založen na modulární architektuře, na nezávislých a nahraditelných komponentách, které zajišťují rozšiřitelnost a udržovatelnost systému.

Vizuální modelování

Vytváří systematický přístup k zachycení požadavků a návrhu systému pomocí grafických prvků a schémat. Specifikuje a dokumentuje systémovou architekturu. Jako standardní jazyk pro modelování slouží „*Unified Modeling Language“* (UML).

Kontrola kvality software

Průběžné ověřování kvality v průběhu celého životního cyklu, identifikace rozdílů mezi návrhem, implementací a zadáním.

Řízení změn

Slouží ke kontrolovanému zapracovávání změnových požadavků čímž zabraňuje nekontrolovatelnému rozšiřování rozsahu projektu a pomáhá identifikovat dopady změn.

Proces vývoje software lze popsat v rámci dvou dimenzí neboli os. Vertikální osa představuje statické hledisko procesu, popis činností, pracovníků, artefaktů a pracovních toků. Horizontální osa reprezentuje dynamický pohled na proces, který je vyjádřen cykly, fázemi, iteracemi a milníky. [11]

Horizontální osa představuje 4 tzv. fáze vývoje [11]

- Počáteční fáze (Inception)
- Elaborační fáze (Elaboration)
- Konstrukční fáze (Construction)
- Fáze nasazení (Transition)

Cílem počáteční fáze je definice požadavků, cílů projektu, vytvoření harmonogramu projektu, sestavení plánu iterací, odhad nákladů a definice rizik projektu. Cílem elaborační fáze je definování architektury systému. Měl by zde být sestaven prototyp, který ověří veškeré architektonické principy a umožní zpřesnit plán realizace systému. V této fázi se také definují komponenty pro opakované použití. V konstrukční fázi probíhá návrh realizace systému včetně testování. Preferuje se zde pokud možno paralelní vývoj. Fáze nasazení zajišťuje nasazení systému do provozu. Součástí bývá školení uživatelů, předání dokumentace atd. Každá fáze je uzavřena milníkem, ve které se kontroluje, zda byly splněny cíle dané fáze a rozhoduje se vstupu do další fáze. [11]

Vertikální osa představuje 9 tzv. disciplín: [13]

- Obchodní modelování
- Požadavky
- Analýza a návrh
- Implementace
- Testování
- Zavedení
- Konfigurační management
- Řízení projektu
- Prostředí

Metodika RUP rozděluje disciplíny na takzvané disciplíny technické a disciplíny podpůrné. Disciplíny obchodní modelování, požadavky, analýza a návrh, implementace, testování a zavedení se řadí do disciplín technických. Tyto disciplíny mohou evokovat jednotlivé fáze vodopádového modelu, je však důležité se uvědomit, že metodika RUP je metodiku iterativní, ve které se opakuje všech 9 disciplín v každé iteraci s různým důrazem a intenzitou.

Disciplíny konfigurační management, řízení projektu a prostředí řadí metodika mezi disciplíny podpůrné.

V roce 2003 došlo k akvizici společnosti Rational Software – původního tvůrce metodiky RUP společností IBM. Popis jednotlivých disciplín metodiky, byl proto převzatý z oficiálních dokumentů společnosti IBM, která se stará o její další vývoj. [13]

Obchodní modelování

Jedním z problémů během životního cyklu vývoje softwaru bývá špatná komunikace mezi byznysovým a vývojovým týmem, to se snaží metodika RUP řešit tím, že těmto dvěma komunitám definuje jednotný jazyk pro modelaci systémů.

Požadavky

Cílem této disciplíny je popsat co nejpřesněji to, co má systém dělat. Důležitým aspektem při tvorbě požadavků je spolupráce dodavatele se zákazníkem, což vede k vzájemnému pochopení potřeb a k požadovaným výsledkům.

Analýza a návrh

Úkolem této disciplíny je znázornit, jak bude systém realizován v implementační fázi. Výsledkem je systém, který:

- provádí funkce a úkoly uvedené v případech užití
- splňuje všechny své požadavky
- je strukturován tak, aby byl komplexní (snadno se upraví, pokud se změní jeho funkční požadavky)

Implementace

Účelem implementace je:

- pokud jde o implementační subsystémy, tak organizovat kód do jednotlivých vrstev
- implementovat třídy a objekty z hlediska komponent (zdrojové kódy, binární soubory, spustitelné soubory atd.)
- provádět jednotkové testy
- implementovat výsledky do spustitelného kódu

Systém je realizován implementací komponent, metodika RUP popisuje, jak opětovně využívat existující komponenty nebo jak implementovat nové komponenty, tak aby byl systém lehčí na údržbu a aby se zvýšila možnost je opětovného použití v dalších iteracích vývoje.

Testování

Tato technická disciplína RUP metodiky má za účel:

- ověřit interakci mezi objekty
- ověřit správnou integraci všech komponent systému
- kontrolovat, jestli bylo dosaženo všech požadavků
- identifikovat a opravit chyby před nasazením softwaru

Jak bylo již uvedeno metodika RUP využívá iterativního přístupu, což znamená, že proces testování probíhá během celého vývoje projektu. Tím se výrazně zvyšuje objevení chyb již v rané fázi vývoje, což snižuje náklady na opravu chyb. Na základě testovacích strategií je definováno, kdy a jak začít s automatizovanými regresními testy, které ověřují stávající funkčnosti systému po implementaci nových změn.

Zavedení (nasazení)

Disciplína se zabývá úspěšným nasazením systému do provozu včetně:

- nasazování externích verzí systému
- vytvoření instalačního balíčku
- distribuování softwaru
- poskytování podpory uživatelům systému
- plánování a provádění beta testů
- migrace stávajícího řešení nebo dat
- akceptace

Přestože je zavedení nové verze systému zaměřeno většinou na konec fáze nasazení, aktivity spojené s tímto procesem jsou zahrnuty již v dřívějších fázích s cílem připravit se na tuto činnost.

Konfigurační management

Tento pracovní postup se zaměřuje na předcházení problémů řízením artefaktů – reálných produktů, které projekt produkuje nebo věcí, které projekt ve fázi vývoje spotřebovává. Je to řízení vstupů, které pracovníci používají pro své činnosti nebo to jsou výstupy, které vznikají činnostmi pracovníků.

Konfigurační management poskytuje metodické pokyny pro řízení paralelního vývoje a pro automatizaci procesů, které jsou důležité a v iterativním vývoji využívané v podstatě na denní bázi. Rovněž popisuje, jak dokumentovat jednotlivý vývoj artefaktů v čase, včetně změn. Definuje proces řízení změnových požadavků včetně, reportingu a řešení nalezených chyb.

Řízení projektu

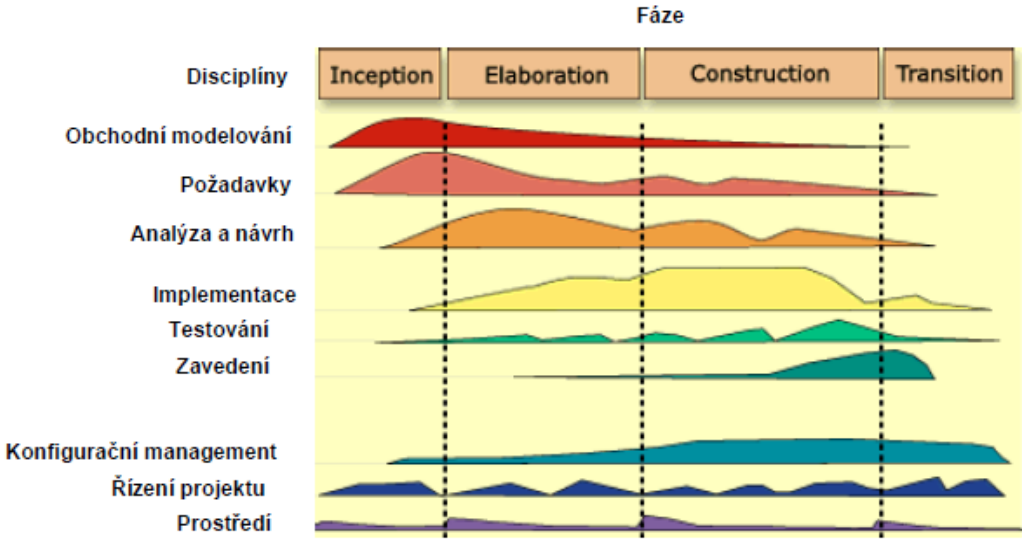
Řízení projektu znamená v podstatě dodávat produkt, který splňuje požadavky a potřeby zákazníků. Řízení projektu se především zaměřuje na specifický aspekt iterativního vývojového přístupu a to tak, že se snaží poskytovat:

- framework pro správu softwarově náročných projektů
- praktické pokyny pro plánování, personální řízení, monitorování a řízení projektu
- framework pro řízení rizik projektu

Prostředí

Hlavním cílem je poskytnou vývojové prostředí a nástroje, které jsou potřebné pro vývoj systému. Tato disciplína se zaměřuje na aktivity spojené s tímto procesem, součástí této disciplíny je i tzv. Development Kit, což jsou další rozšíření RUP metodiky pro implementování specifických požadavků organizace či projektu.

Obrázek č. 4 znázorňuje jednotlivé fáze a disciplíny metodiky RUP. Jak bylo zmíněno dříve, proces testování probíhá s různou intenzitou v každé fázi vývoje.



Obrázek 4 - Fáze a disciplíny RUP [11]

3.3.2 Extrémní programování (XP)

Extrémní programování je agilní metodika, kterou vytvořil s dalšími spolupracovníky Kent Beck. Jedná se o metodiku, ve které jsou běžné činnosti vývoje softwaru dovedeny do extrému. Tím by mělo být zajištěno to, že se vývoj dokáže přizpůsobit často měnícím se požadavkům během vývoje a dodávat tak software vyšší kvality: [11]

- jestliže se osvědčují revize kódu, kód se bude neustále revidovat (párové programování)
- pokud se osvědčuje testování, vývojáři budou neustále psát jednotkové testy (testování jednotek), testování bude probíhat i na straně zákazníka (akceptační testy)
- jestliže se osvědčuje návrh, refaktorování se stane každodenní součástí
- pokud se osvědčuje jednoduchost, bude se volit to nejjednodušší možné řešení
- osvědčuje-li se důležitost architektury, bude se neustále dovytvářet a definovat (metafora)
- pokud se osvědčuje testování integrace, bude se integrovat a testovat několikrát denně (průběžná integrace)
- jestliže se osvědčují krátké iterace, bude se plánovat v opravdu krátkých iteracích: vteřiny, minuty, hodiny nikoli týdny, měsíce či roky. (plánovací hra)

Extrémní programování dodržuje 6 základních pravidel neboli postupů vývoje: [15]

Zadání

Na začátku projektu dochází k sepsání tzv. *„User stories“* – uživatelských příběhů (scénářů), definuje se seznam funkčních požadavků a akceptačních kritérií. Akceptační kritéria může zákazník kdykoliv doplňovat, či upravovat, avšak za cenu restartu cyklu vývoje, což má mnohdy i peněžní dopady.

Plánování

Cílem této disciplíny je sestavení časového harmonogramu projektu, projekt se rozdělí do jednotlivých iterací. Každá iterace pak začíná novým plánováním. Dochází k častým vydáváním malých změn. V této fázi se také měří rychlost vývoje pro další analýzu.

Design

Pro každý systém musí existovat metafora, cení se jednoduchost řešení. Žádná funkčnost systému se nepřidává předčasně, k přidání dochází právě tehdy, kdy je jí zapotřebí. Dochází k častému refaktorování kódu, kdykoliv a kdekoliv je potřeba.

Programování

Zákazník musí být v této fázi vždy k dispozici. Na začátku programování se vždy píší jednotkové testy. Extrémní programování využívá praktiky párového programování. Zdrojové kódy vlastní všichni programátoři, každý programátor přispívá a zodpovídá za program. Optimalizace kódu se provádí až v konečné fázi. Pracovní doba je 40 hodin týdně, nevyžadují se žádné přesčasy.

Testování

Jak bylo řečeno každý kód musí mít své jednotkové testy, kterými musí úspěšně projít, než je nasazen. Kdykoliv je nalezena chyba, jsou na ni napsány další testy, které zabraňují jejímu opakování v dalších fázích vývoje. Během iterace se plánuje, z jakých uživatelských příběhů se napíší akceptační a regresní testy. Nad aplikací pak probíhají tzv. black-box testy jejichž výsledky ověřuje zákazník, který určí, prioritu chyb v testech, které neprošli. Na opravu těchto chyb se pak vývoj zaměří v další iteraci. Uživatelský příběh není považován za kompletní, dokud nejsou úspěšně splněny všechny akceptační testy. Z důvodů častého provádění těchto testů v jednotlivých iteracích, by se měli tyto testy automatizovat.

Dodání a akceptace

Dodavatel projektu má povinnost dodat zákazníkovi akceptační prostředí, nasadit do něj projekt a případně namigrovat stávající data. Na základě funkčních požadavků a akceptačních kritériích definovaných ve fázi zadání, provádí zákazník akceptační testy. Pokud dodávka splňuje všechna akceptační kritéria má zákazník povinnost projekt převzít.

Může se však stát, že zákazník doplní, některá akceptační kritéria a inicializuje tak další část vývoje, samozřejmě placenou.

Metodika extrémního programování je založena na 5 základních hodnotách, jak říká Kent Beck, nejsou to úplně pravidla, ale spíš hodnoty, které by se měly dodržovat pro harmonický chod: [16]

Komunikace

Klíčem celého procesu vývoje je komunikace, každý člen je součástí týmu, komunikace probíhá na denní bázi, společně se spolupracuje od sběru požadavků až po samotné programování aplikace.

Jednoduchost

Klade se důraz na jednoduchost, vyvíjeno je jen to, co je požadováno, nic víc. Malými krůčky se dopracovává k výslednému řešení s minimalizací selhání.

Zpětná vazba

Dochází k časté demonstraci výsledků zákazníkovi, každá připomínka se bere vážně. Pečlivě se naslouchá požadovaným změnám, přizpůsobuje se projekt, ne proces.

Respekt

Každý je respektován, je brán jako plnohodnotný člen týmu. Každý přispívá k projektu i když by to byl jen entusiasmus. Vývojáři respektují připomínky zákazníka a naopak.

Odvaha

Hlavně u programátorů je zapotřebí odvahy, pokud se například pustí do opravy chyby, která může vyvolat kaskádu dalších souvisejících problémů.

Extrémní programování využívá 12 praktik, které se dělí do tří základních oblastí: [17]

Plánovací hra

Byznys a vývoj spolu neustále komunikují s cílem dosáhnout maximálního ekonomického přínosu za co nejmenší čas. Plánuje se v různých měřítkách, ale základní pravidla jsou vždy stejná:

- byznys přichází se specifikací zákaznických požadavků, přepsaných do uživatelských příběhů
- vývoj odhaduje kolik úsilí ho bude stát vývoj daného příběhu a kolik času mu z jednotlivé iterace může věnovat
- byznys rozhoduje jaké funkce se budou implementovat nejdřív a jak často bude docházet k uvolňování nových verzí systému

Vydávání malých verzí

Brzy a často se vydávají malé verze systému s novou funkcionalitou.

Metafora

Jsou zavedeny konvence, popisy, jména, aby každý člen týmu dokázal jednoduše komunikovat o částech systému.

Jednoduchý návrh

Vždy se používá ten nejjednodušší návrh ke splnění požadavků. Požadavky se často mění, s jednoduchým návrhem se jim pak není těžké přizpůsobit.

Průběžné testování

Pro každou funkčnost nejdřív vývojáři píší jednotkové testy, až potom ji vyvíjí. Implementace probíhá až po splnění všech testů:

- jednotkové testy jsou automatizované testy psané přímo vývojáři, testují jednotlivé části kódu, často jednotlivou třídu nebo skupinu tříd.
- Akceptační testy (funkční testy) typicky testují celý systém nebo jeho větší část

Refaktorování kódu

Zdrojové kódy se neustále refaktorují, aby se udržela jejich jednoduchost a přehlednost, díky jednotkovým testům je zaručeno, že zásahem do kódu nedojde ke ztrátě funkčnosti.

Párové programování

Na produkčním kódu vždy spolupracují dva programátoři, jeden z nich píše kód, druhý přemýšlí o souvislostech nebo jiných řešeních. Okamžitou komunikací vzniká rychlá zpětná vazba.

Společné vlastnictví kódu

Žádný vývojář není vlastníkem konkrétního modulu, od každého vývojáře se očekává, že bude schopen kdykoliv pracovat na jakémkoli modulu.

Průběžná integrace

Často, minimálně jednou denně dochází k implementování nových změn do produkčního kódu, před i po integraci dochází k testování, z důvodů nalezení případných chyb.

Udržitelné tempo

Pracuje se 8 hodin denně, 5 dní v týdnu, přesčasy jsou v extrémním programování známkou špatně nastaveného, nefungujícího procesu.

Skutečný zákazník

Musí být zajištěna komunikace mezi vývojem a skutečným zákazníkem, což je uživatel, který bude výsledný produkt používat. U komerčních systémů s mnoho uživateli tuto roli zastupuje produktový manažer.

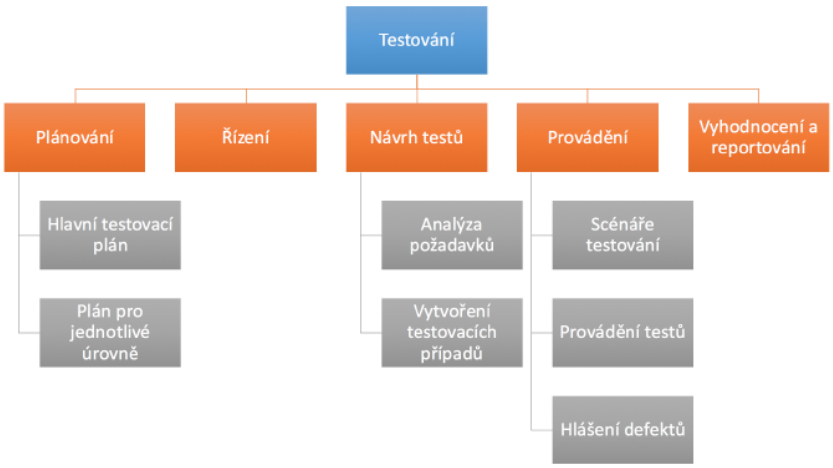
Standardizace kódu

Důležitým předpokladem pro společné vlastnictví kódu je to, že všichni programátoři dodržují stejné konvence při jeho psaní. Standardizace je důležitým prvkem při párovém programování a při refaktorování kódu.

3.4 Testování softwaru

V této kapitole bude vyčleněn samotný proces testování, vymezení procesu testování jako pojmu, významu a jeho základních principů.

Jednoznačně definovat testování jako proces není jednoduché, mnoho starších definic uvádí, že testování je proces, který má za cíl nalézt chyby v softwaru. S tím bych si dovolil nesouhlasit, jelikož nalezení chyb je pouze podmnožinou aktivit, které testování skýtá. Mimo tuto aktivitu existují i aktivity před a po vykonáváním testů, jako například plánování testů, návrh testovacích případů, vyhodnocení testů nebo tvorba reportů o probíhajícím testování.



Obrázek 5 - Aktivity v procesu testování [19]

Častým omylem se kterým se autor setkal je zaměňování pojmu testování s pojmem zajišťování kvality (Software Quality Assurance). Zajišťování kvality softwaru je proces zaměřený na zajišťování kvality celého životního cyklu vývoje software včetně plánování, na rozdíl od toho řízení kvality softwaru (Software Quality Control) se zaměřuje přímo na výstupy z jednotlivých procesů, u kterých se ověřuje, zda odpovídají specifikacím a požadavkům. [18]

Testování z pohledu návrhu testovacích případů, jejich exekuce a následného vyhodnocení je definováno jako:

„Proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatelů.“ [18]

Jedná se o řízený proces, neboť vše je prováděno s určitým úmyslem, předchází mu plánování a vyhodnocení testů.

Boris Beizer rozděluje testovací proces do 5 úrovní podle jeho vyspělosti: [20]

- Úroveň 0: Nerozlišuje se testování a ladění, které provádí vývojáři. Testování je chápáno jen jako aktivita pomáhající odstranit chyby.
- Úroveň 1: Testování softwaru má prokázat jeho funkčnost, je zaměřeno na prezentaci softwaru.
- Úroveň 2: Testování softwaru má prokázat, že software nefunguje. Cílem je nalézt defekty a odlišnosti oproti specifikaci.
- Úroveň 3: Testování je zaměřeno především na snížení rizika. Proces testování je rozšířen o aktivity v průběhu celého životního cyklu vývoje softwaru.
- Úroveň 4: Testování a ostatní aktivity řízení kvality jsou chápány jako aktivity preventivní. Testování se systematicky zaměřuje na předcházení chyb ve všech fázích vývoje.

3.4.1 Verifikace a validace

Podle klasického přístupu se dají tyto dva pojmy rozlišit, pokud si položíme dvě základní otázky. Verifikace znamená, pokud se ptáme: „Vytváříme produkt správně?“, zatímco validace znamená položit si otázku: „Vytváříme správný produkt?“. [6]

Verifikace má za cíl ověřit, zda software splňuje požadavky dané specifikace. Souvisí s formálními aspekty vývoje.

Validace se provádí za účelem zjištění, zda software vyhovuje požadavkům uživatelů.

3.5 Chyba

Proces vývoje softwaru a samotný software je výsledkem spolupráce systémových architektů, analytiků, vývojářů a dalších specialistů, kteří mohou v určité míře ve své práci chybovat. Následkem těchto pochybení je potom zavlečení chyb do jednotlivých procesů vývoje nebo do samotného softwaru. [18]

Defekt v procesu vývoje vzniká pochybením člověka a je příčinou chyby. Chyba je určitý nežádoucí stav systému, který může vést k selhání. Selhání je následný nesoulad mezi specifikací a aktuálním chováním systému. [18]

O chybě hovoříme, pokud je alespoň jedno z těchto 5 tvrzení pravdivé: [2]

1. Software nedělá něco, co by dle specifikace dělat měl
2. Software dělá něco, co by dle specifikace dělat neměl
3. Software dělá něco, co není specifikováno
4. Software nedělá něco, co není specifikováno, ale mělo by být
5. Software je těžko srozumitelný, těžko ovladatelný, pomalý nebo vykazuje jiné vlastnosti bránící jeho efektivnímu použití

3.5.1 Závažnost a priorita defektu

Nalezené defekty mají na systém různý vliv, proto se při reportování defektů určuje jejich závažnost (severity) a stupeň naléhavosti označovaný jako priorita (priority).

Neexistuje žádná obecná škála těchto dvou klasifikací defektů. Každá organizace si ji nastavuje sama podle specifičnosti systému a úrovně testování.

Často se však používá základní 4 stupňová škála podle závažnosti defektu: [18]

- Kritická: Kritická chyba systému, daná funkcionalita nefunguje, neexistuje workaround, nedá se pokračovat.
- Vysoká: Chyba ovlivňuje významné funkce systému, existuje workaround, systém je možné v omezené míře používat.
- Střední: Chyba neovlivňuje významné funkce systému, systém je možné s obtížemi používat.
- Nízká: Většinou se jedná o drobné nedostatky typu kosmetických chyb.

Při určování naléhavosti defektu se často užívají 4 úrovně priority: [18]

- 1 – Kritická: Nejvyšší důležitost, není možné pokračovat v testování nebo vývoji, blokační defekt, opravu je nutné provést okamžitě.
- 2 – Vysoká: Částečně blokuje testování nebo vývoj, oprava je nutná hned jak to bude možné.
- 3 – Střední: Bez dopadu na proces testování, oprava je prováděna v rámci běžného pořadí opravy defektů.
- 4 – Nízká: Nejnižší priorita, oprava defektů přichází na řadu až po opravě důležitějších defektů, někdy bývá oprava odložena do dalších kol nebo k opravě vůbec nedochází.

3.5.2 Zaznamenávání chyb

Mezi další činnosti testera, které v rámci procesu testování, mimo navrhování, plánování a exekuci testů vykonává, je zaznamenávání nalezených chyb. V rámci celého procesu se jedná o jednu z nejdůležitějších činností vůbec. [21]

Z tohoto důvodu by měl být kladen určitý důraz na správné zaznamenávání nalezených chyb. Mnohdy se stává, že nalezené chyby nejsou opraveny, částečně to může být tím, že byly vedením projektu označeny jako nedůležité, jejich oprava je v současné fázi riskantní nebo se je opravit z nějakého důvodu nepodařilo. Tyto důvody tester nemůže ovlivnit.

Stává se však i to, že některé důležité chyby nejsou opraveny z důvodů jejich špatného zadání a následného nepochopení vývojářem. V tomto případě nese za vzniklou situaci částečně zodpovědnost tester. [22]

To znamená, že by při nalezení chyby měli být dodržována určitá pravidla: [22]

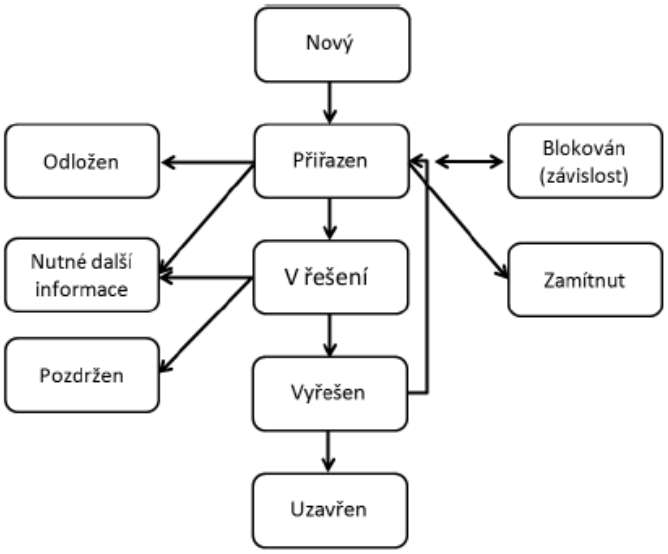
- nalezené chyby co nejdříve reportovat
- nalezené chyby účinně popsat
- sledovat stav nalezených chyb

Předpokladem po rychlé a účinné opravení chyby je tedy její řádné zaznamenání, popis chyby by měl proto být: [21]

- Minimální – je nutné, aby popsané chyby obsahovaly informace k jejich opětovnému vyvolání, měl by být uveden postup – sled kroků k nasimulování
- Jednotlivý – tester by měl reportovat chyby samostatně, pokud bude popisovat v jednom reportu více chyb, které vznikly například kaskádou, může se stát, že vývojář opraví jednu chybu a ostatní přehlédne
- Jasný a reprodukovatelný – má-li být chyba opravena, je nutné, aby popis obsahoval všechny důležité informace k jejímu nasimulování jako je vývojové prostředí, kde byla chyba nalezena, použitá data, čas, zkrátka vše, co by mohlo vývojáři pomoc s identifikací problému

Jak již bylo zmíněno zaznamenáním chyby úloha testera nekončí, je nutné sledovat a správně nastavovat její aktuální stav, aby nedocházelo ke zdržování v odbavování defektů.

Na obrázku č. 6 je znázorněn model životního cyklu defektu s možnými přechody do jiných stavů.



Obrázek 6 - Životní cyklus defektu 1 [18]

3.6 Role v testovacím týmu

Jak již bylo uvedeno v kapitole 3.4 Testování softwaru, proces testování není pouze o provádění testů, jedná se o komplexní proces zahrnující celou řadu činností. Proto se podle potřeb projektu sestavuje různě velký tým specialistů, kteří zastávají specifické role. Struktura týmu může být podle potřeb organizace, či konkrétního projektu různá. Obecně jsou však v testovacím týmu zastoupeny následující role.

3.6.1 Manažer testování (Test manager)

Manažer testování je v testovacím týmu na pomyslném vrcholu pyramidy. Je to osoba, která zodpovídá za celý proces testování a jeho realizaci. Je odpovědný za aktivity spojené s testováním, které reportuje vyššímu managementu. Jeho obvyklou rolí je volba přístupu k testování, nastavení cílů testování, volba strategie testování. Vyjednává s managementem, vývojovým týmem či zákazníkem. Řeší vzniklé problémy, které by mohly narušit proces testování, jako je například nefunkční prostředí. Mezi jeho pravomoci patří i řízení personálních zdrojů, rozhoduje o podobě testovacího týmu nebo jeho změnách v průběhu projektu. Zajišťuje pro svůj tým potřebné nástroje k testování. [18]

3.6.2 Vedoucí testování (Test Lead)

V testovacím týmu je úkolem vedoucího testování, v souladu s testovací strategií, vytvoření plánu testování a kontrola jeho dodržování. Koordinuje ostatní členy týmu, dohlíží na plnění jejich úkolů, spravuje systém pro zaznamenávání chyb. Tvoří pravidelné reporty o stavu testování, nalezených chybách atd. Často se zapojuje i do samotného procesu návrhu či exekuce testů. [18]

3.6.3 Analytik testování (Test analyst)

Úkolem analytika testování je důkladná analýza požadavků na jejichž základě definuje testovací případy. Ze získaných testovacích případů definuje jednotlivé pozitivní a negativní testovací scénáře. Dalším krokem je pak samotný návrh testovacích scénářů se všemi předpoklady k testování. U scénářů stanovuje riziko a určuje jejich prioritu k otestování. Často také vytváří sady regresních testů pro ověření stavu systému po nasazení nových změn. Tato role bývá často spojena s rolí testera, z čehož vyplývají další povinnosti.

3.6.4 Tester (Tester)

Tester je zodpovědný za provádění testů připravených analytikem testování. Zaznamenává nalezené chyby a dál sleduje jejich životní cyklus. Po opravě chyby provádí její přetestování a případnou aktualizaci stavu testovacího případu. Další činností, kterou tester provádí, je příprava a údržba testovacích dat potřebných pro testování.

Na projektech, které využívají automatizace v testování se tato role dále rozděluje na specialisty automatizovaného testování. Tito členové spolupracují s architekty systémů a analytiky, dle technologií a potřeb testování volí testovací nástroje a automatizují testovací scénáře. Jejich úkolem je tvorba, a především údržba automatizačních scriptů. Zaznamenávají a reportují jednotlivé běhy testů, tvoří potřebnou dokumentaci. Často spravují vývojové prostředí, na kterém testy provádí. [18]

Automatizované testy se používají hlavně při regresním testování, kdy je potřeba otestovat, zda byla zachována stávající funkčnost systému po nasazení nových funkcionalit nebo opravení defektů. Další oblastí, kde se využívá automatizace v testování, jsou zátěžové testy.

3.7 Typy testů

Testy se dělí podle různých hledisek, cílem této kapitoly bude vyčlenění typů testů podle účelu testování.

Funkční testy

Tento typ testů je zaměřen na funkcionalitu systému, ověřuje se, zda aplikace splňuje veškeré specifikované funkční požadavky, které jsou implementovány. Funkční testy mají významný vliv na výslednou bezchybnost systému, proto je na tuto oblast testování kladen velký důraz. V porovnání s ostatními typy testů bývá funkčními testy nalezeno nejvíce chyb. [23]

Bývá pravidlem, že na konci fáze integrace, tedy před vstupem do fáze systémového testování, se provádí takzvané smoke testy. Tento typ testů nemá za úkol odhalovat chyby, ale zkoumá stabilitu a připravenost systému před dalším funkčním testování, testují se hlavní části systému, výsledky testů poskytují informaci o tom, jestli je možné pokračovat v testech.

Nefunkční testy

Princip těchto testů spočívá v testování nefunkčních požadavků, jak z názvu vyplývá. Jedná se tedy o testy, které přímo nesouvisí s funkcemi, ale zároveň testují podstatné vlastnosti softwaru, které mají vliv na správné fungování. Existuje mnoho typů nefunkčních testů, zmíněny však budou dvě hlavní oblasti.

Performance testy neboli výkonové testy, které mají za úkol ověřit, zda dokáže aplikace spolehlivě fungovat při zvýšeném počtu současně pracujících uživatelů. Spolu s performance testy se často provádí i load testy, taktéž označované jako testy zátěžové. Na rozdíl od stress testů, které mají za úkol odhalit možné kritické chyby související s vysokou zátěží aplikace, performance a load testy se provádí za účelem optimalizace.

Security testy neboli bezpečnostní testy jsou dalším typem nefunkčních testů. Mezi tyto testy se řadí především testy penetrační. Penetrační testy spočívají v simulaci útoků na aplikaci s cílem odhalení slabých míst. [23]

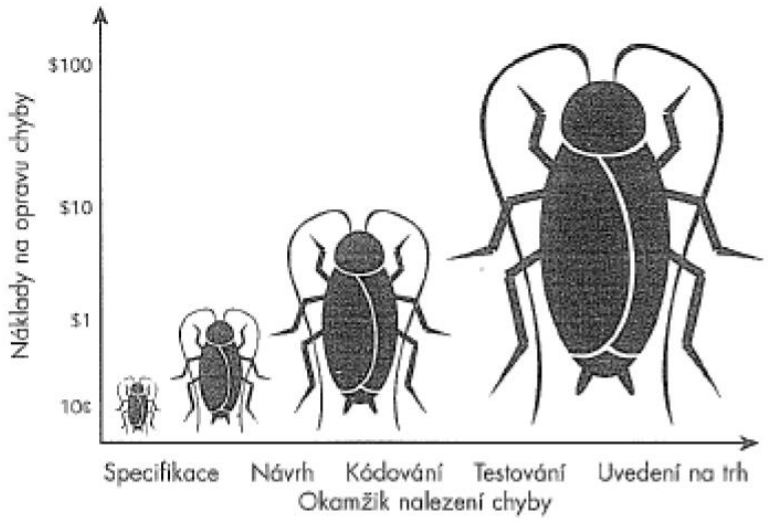
Regresní a konfirmační testy

Regresní testy se provádí za účelem ověření, že oprava chyby nebo implementace nové funkčnosti nezpůsobila nesprávné fungování dosud fungujících částí aplikace. Z pohledu časové náročnosti a množství testů je prakticky nemožné otestovat vždy celý systém po nasazení těchto změn, proto se při regresních testech definují kritické oblasti, které by mohly být změnou ohroženy a na ty je zaměřeno testování. U těchto typů testů je často využívá automatizace.

Testy zabývající se ověřením správnosti konkrétní opravy nalezené chyby se nazývají konfirmační testy. [6]

3.8 Techniky testování

Předpokladem pro vývoj kvalitního softwaru je jeho testování po celou dobu vývoje. Jednak protože by mohlo velké množství nalezených defektů v pozdějších fázích vývoje zcela ochromit další vývoj, tak i proto, že náklady na opravu defektu jsou nižší podle toho, čím dříve je defekt objeven. [2]



Obrázek 7 - Náklady na opravu defektu v závislosti na čase [2]

Existuje celá řada technik vzniklých s cílem předcházet těmto skutečnostem. Tato kapitola se bude zabývat jejich členěním.

3.8.1 Techniky testování podle způsobu provedení

Podle způsobu provedení se testování dělí nejčastěji na testování černé skříňky tzv. black-box testing a na testování bílé skříňky neboli white-box testing. Tento přístup vychází z úrovně znalosti testované aplikace, kterou mají analytici testování a samotní testeři aplikace.

Testování černé skříňky vychází z předpokladu, že testeři nemají znalost kódu aplikace a ani jejího vnitřního fungování. Tento způsob testování je výhodný hlavně při simulaci reálného použití aplikace, jelikož koncoví uživatelé neznají nebo nepotřebují znát, jak byla aplikace navržena či naprogramována. Tento přístup s sebou však nese riziko nadbytečného testování některých částí, a naopak nedostatečného testování částí jiných.

Oproti tomu přístup založený na znalosti fungování aplikace, jejích vnitřních principů a analýze kódu se nazývá testování bílé skříňky. Testovaní touto techniku je velice důsledné, avšak s realistickými scénáři užití se míjí.

Toto dilema, jakou zvolit techniku částečně řeší tzv. testování šedé skříňky, známé pod pojmem gray-box testing. Testy jsou navrhovány ze zákaznicky orientovaného pohledu, to znamená stejně jako u černé skříňky, následně se ale využívá postupů testování bílé skříňky k dosažení pokrytí všech částí aplikace. Je tedy důležité mít nadhled a volit testy, jak podle užitečnosti, tak podle toho, aby byla zajištěna bezchybnost aplikace. [24]

Speciální oblast testování tvoří tzv. ad hoc a exploratorní testování. Testování ad hoc je prováděno nesystematicky, bez patřičného plánování a nahodile. Cílem je nalezení co nejvíce chyb. Exploratorní testování je systematicky pojaté ad hoc testování, kdy tester nejdříve zkoumá aplikaci, snaží se zmapovat její funkce a data. Na základě zkoumání poté identifikuje rizikové oblasti a volí strategii testování. Tato technika je silně založena na zkušenostech a znalostech testera. [18]

3.8.2 Techniky testování podle úrovně vývoje

Během vývoje je testování prováděno na různých úrovních. Systém může být testován jako celek nebo se může testování zaměřit na jeho jednotlivé části. Obrázek č. 2 tyto úrovně či fáze ilustruje. Techniky testování jsou tím pádem voleny podle aktuální úrovně vývoje a dělí se na:

- Jednotkové testování
- Integrační testování
- Systémové testování
- Akceptační testování

Jednotkové testy

Někdy označované jako testování komponent, jedná se o testování samostatných modulů, v objektově orientovaném přístupu nejčastěji metod a tříd. Testy jsou psané formou programového kódu samotnými vývojáři. Pro vytváření testů se využívá již existujících specializovaných frameworků, v jazyce Java například frameworku JUnit. Z důvodu složitosti aplikování těchto testů, do již vytvořených aplikací a nutnosti kompletního refaktorování, by se měli vývojáři věnovat jejich návrhu už před samotným vývojem aplikace. [25]

Integrační testy

Cílem integračních testů je ověření komunikace mezi komponentami aplikace. Integrace komponent lze však ověřovat nejen mezi sebou, ale i mezi komponentou a systémem, hardwarem či dokonce mezi komponentou a rozhraním jiného systému. U těchto testů platí, že čím je větší rozsah integračních testů, tím je složitější lokalizovat chyby určitého systému. Při testech by nemělo docházet k testování samotné funkcionality systémů, nýbrž k testování integrace mezi nimi. [25]

Systémové testy

Po integračních testech, tedy po ověření integrace systému přicházejí z pravidla na řadu testy systémové. V této fázi se testuje aplikace jako funkční celek. Aplikace je testována z pohledu zákazníka, kdy se simulují kroky, které by mohly v praxi nastat. Obvykle probíhá několik kol těchto testů, ve kterých jsou prováděny funkční i nefunkční testy, opravy defektů v jednotlivých kolech jsou samozřejmě reportovány a po opravě přetestovány v dalších kolech. Jedná se o poslední fázi testů před finálním předání zákazníkovi, dalo by se říct, že tyto testy slouží jako výstupní kontrola produktu.

Někdy se spojují systémové testy s integračními, takové testy se pak nazývají jako fáze SIT – fáze systémově-integračních testů. [25]

Uživatelské akceptační testy

Po úspěšně ukončené fázi systémových testů je aplikace připravena na nejdůležitější validační aktivitu – akceptační testování samotným zákazníkem. Smyslem je ověření, zda aplikace splňuje akceptační kritéria, jinak řečeno, jestli splňuje měřitelné a ověřitelné podmínky pro přijetí produktu. Akceptační testy provádí testovací tým zákazníka podle připravených akceptačních scénářů. Nalezené nesrovnalosti ve specifikaci, ale například i v dokumentaci a dalších součástí produktu zákazník reportuje zpět vývojovému týmu. [25]

V této fázi se využívá především end-to-end testování (zkráceně E2E). Princip tohoto typu testování spočívá ve sledování určité entity po celou dobu její životnosti napříč systémem. Jako příklad by se dalo uvést přihlášení uživatele do aplikace, zobrazení dat, provedení nějaké změny a jeho odhlášení. [18]

3.9 Testovací případy a základy jejich návrhu

Norma IEEE 1012:2004 definuje testovací případ jako sadu vstupů, podmínek pro spuštění a očekávaných výsledků, vyvinutou za účelem provedení specifických kroků programu nebo ověření souladu s konkrétním požadavkem. Jinými slovy, testovací případ ověřuje, zda systém za daných podmínek reaguje na vstupy v souladu se specifikací.

Testovací případ je tvořen posloupností kroků. Většinou je testovací případ definován dvojicí akce kontra očekávaná reakce systému, tato dvojice tvoří jeden konkrétní krok v celém testovacím scénáři. [18]

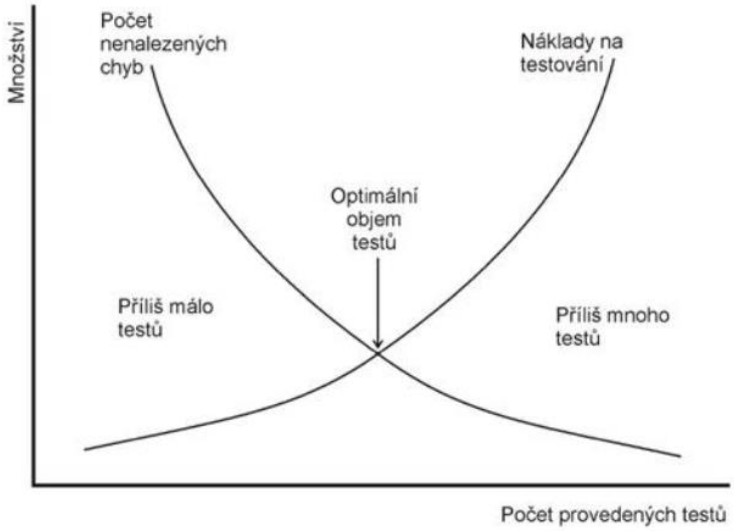
Podle zmíněné normy by měl testovací případ obsahovat následující strukturu:

- unikátní identifikátor testovacího případu
- zaměření nebo účel testu
- specifikace dat potřebných k provedení testu
- specifikace vstupních hodnot, očekávaného chování a vlastností
- potřeby na prostředí, software, hardware
- prerekvizity pro splnění testu
- případné závislosti na jiných testovacích případech nutných ke splnění testu

Tato forma zápisu se však často nevyužívá, neboť v praxi existují, namísto takto vytvářených textových dokumentů, efektivní nástroje pro správu testů a report nalezených chyb, které navíc integrují další funkcionality pro podporu procesu testování. [18]

3.9.1 Optimální objem testovacích scénářů

Vyčerpávající testování je nemožné, s rostoucím počtem provedených testů ve procesu testování produktu, sice klesá pravděpodobnost nalezení chyby, je však nutné si uvědomit, že s rostoucím počtem testů roste i časová náročnost a zvyšují se náklady na testování. Objem testů je proto nutné plánovat v souladu s časovým a rozpočtovým plánem projektu. [2]



Obrázek 8 - Optimální objem testovacích případů [26]

3.9.2 Návrh testovacích případů podle struktury programu

Testování pokrytím příkazů

Testy jsou navrhovány, tak aby došlo alespoň jednou ke spuštění všech příkazů. Eliminace určitých typů chyb je zajištěna plným pokrytím příkazů. Tím pádem je zajištěno, že v kódu neexistují žádné neotestované příkazy, to ovšem neznamená, že bude kód dobře otestován. Tato technika obsahuje mnoho nedostatků, její význam je dosti okrajový. [18]

Testování pokrytím hran či rozhodování

Tato technika se zabývá pokrýváním všech stavů, které mohou nastat při větvení programu. Často se užívá termín pokrývání hran, to je odvozeno z reprezentace programu pomocí grafu. Program je znázorněn jako řízený tok, přechod do určité hrany je řízen

rozhodovací podmínkou. Technika návrhu testovacích případů tedy spočívá v pokrytí všech těchto hran neboli možných toků programu. [21]

3.9.3 Návrh testovacích případů podle specifikace

Rozdělení tříd ekvivalence

Testy jsou navrhovány s cílem ověřit, zda systém reaguje na vstupy, tak jak by podle specifikace měl. Základním předpokladem je, že systém reaguje na vstupy ze stejné třídy ekvivalence stejně. Vstupy jsou rozděleny do jednotlivých tříd, testování systému potom probíhá pomocí vybraných prvků z konkrétních tříd. Rozdělování vstupů do tříd ekvivalence je možné aplikovat ve všech úrovních testování. [22]

Analýza hraničních hodnot

Jedná se o rozšíření techniky rozdělování do tříd ekvivalence. Nezabývá se rozdělováním vstupů do tříd ekvivalence, ale zkoumá a identifikuje hraniční oblasti tříd. V těchto oblastech existuje větší pravděpodobnost přítomnosti chyby, testování je tedy soustředěno právě na tyto okrajové hodnoty (minimální a maximální hodnoty tříd). Tento typ návrhu lze však použít jen za podmínek, že dokážeme prvky tříd ekvivalence takto rozdělit. [22]

Testování rozhodovacích tabulek

Testovací případy jsou navrhovány za pomoci rozhodovacích tabulek. Tabulky obsahují všechny kombinace podmínek, vstupů a výstupů. Na začátku návrhu se provádí důkladná analýza specifikace a identifikace všech podmínek a akcí. Na základě získaných informací je sestavena rozhodovací tabulka, která znázorňuje akce systému vyvolané kombinací podmínek. Tento přístup k návrhu testů se využívá především při testování složitých pravidel či byznys logiky aplikace. Bez sestavení rozhodovacích tabulek by byl návrh testů velice obtížný a mohlo by dojít k opomenutí otestování některých akcí systému. [18]

Testování přechodů mezi stavy

Návrh testů je zaměřen na testování stavů určité entity a přechodů mezi nimi. Cílem je potvrzení správnosti přechodu mezi definovanými platnými stavy, případně identifikovat neplatné či nespecifikované přechody. Stav je chápán jako stálá konfigurace všech prvků při čekání na konkrétní událost. Přechody mezi stavy se popisují stavovým digramem nebo tabulkou o 4 sloupcích: *Výchozí stav, Událost, Akce a Nový stav.* [18]

Návrh testovacích případů podle případů užití

Tato metoda se zabývá návrhem testovacích případů odvozených z různých modelů, které znázorňují interakci uživatelů se systémem. Případy užití popisují dosažení cíle nějaké funkcionality jako interakci uživatele a systému. Soubor všech případů užití tedy zachycuje kompletní funkčnost systému. Tato funkční specifikace bývá často součástí specifikace požadavků na software. [27]

Další součást případu užití kromě modelu, často diagramu aktivit, je specifikace případu užití. Specifikace nemá žádnou pevně stanovenou podobu, může být ve formě tabulky nebo v podobě slovního popisu.

Specifikace by však měla obsahovat následující části: [27]

- krátký popis vysvětlující danou funkcionalitu
- aktéry neboli uživatele interagující se systémem v různých rolích, například administrátor nebo běžný uživatel
- podmínky pro spuštění
- základní scénář, jedná se o základní tok programu popsaný jako sekvence kroků vykonaných uživatelem, kde se střídá akce uživatele a reakce systému
- alternativní scénáře, scénáře popisující možné odchylky od základního scénáře, jedná se vždy o alternativu konkrétního kroku v základním scénáři, způsobenou aktérem či systémem
- podmínky pro dokončení

Správně navržené případy užití jsou z pohledu testování ideálním případem pro návrh testovacích případů. Navržené testy vycházejí z reálného použití systému, proto slouží k nejpřínosnějším testům při odhalování chyb.

Návrh testů podle případů užití se používá jak v systémově-integračním testování, tak i při akceptačních testech. [18]

3.9.4 Návrh testovacích případů založený na zkušenostech

Předchozí typy návrhu testovacích případů využívali k návrhu dvou odlišných přístupů. Návrh testů podle struktury programu využívá přístupu bílé skříňky, vychází se z detailní znalosti zdrojového kódu, zatímco u metoda návrhu testovacích případů podle specifikace využívá přístupu černé skříňky. K těmto technikám se řadí další přístup k návrhu testovacích případů, tato technika je založena na zkušenostech testera či analytika testování. Předpoklad pro využití této techniky jsou dostatečné zkušenosti s testováním, znalost dané oblasti testování, ale i širší porozumění celé problematiky. Důležitou vlastností je i technická znalost, která umožňuje pochopit fungování aplikace z pohledu architektury na základě které lze zaměřit testování i na potenciální problémy aplikace z pohledu funkčních i nefunkčních požadavků.

Tento přístup se volí hlavně tehdy, kdy není k dispozici specifikace nebo je ve velmi špatném stavu. Ačkoli se tento přístup k testování založený na zkušenosti analytika nebo testera jeví jako účinný, nemělo by se jednat jako jediný přístup k návrhu testů, případně k testování systému. [18]

3.10 Plánování testování

Obdobně jako v případě projektového managementu nemůže testování probíhat nahodile či neurčitě. Bylo by velice obtížné dodávat kvalitní produkt nebo provádět kvalitní testování, pokud by nebylo specifikováno, co bude implementováno, nebyla by popsaná funkcionalita nebo by testovací tým nedefinoval, co bude předmětem testování a jakých prostředků bude v procesu testování využíváno. Důležitým prvkem testovacího plánu je i časový harmonogram projektu, který je tvořen činnostmi, které jsou promítnuty do celkového časového plánu projektu, což je důležité zejména pokud nastanou nečekané problémy a je zapotřebí zareagovat na vzniklé skutečnosti. [22]

Za tímto účelem jsou vytvářeny plány testování, plány se liší podle požadavků každé organizace nebo podle typu či přístupu k testovanému produktu. Existují různé šablony, které definují podobu plánu testování, avšak hlavním problémem při vytváření plánů testovaní podle šablon je věnování přílišné pozornosti na vytvoření dokumentu, namísto směřování pozornosti na skutečné vytvoření plánu testování, které odpovídá testovanému produktu.

Standard pro dokumentaci testování softwaru definuje testovací plán jako dokument, jehož smyslem je:

„Předepsat rozsah, postup, prostředky a časový plán aktivit spojených s testováním. Identifikovat jednotlivé testované položky, testované funkce a úkoly prováděné při testování, konkrétní osoby odpovědné za každý z úkolu a rizika spojená s definovaným plánem.“ [28]

Jak již bylo řečeno mezi organizacemi je dokumentace k testování různě strukturována, typicky je však v souladu s doporučenými standardy řazena dle hierarchie na: [18]

- Zásady testování, které popisují filozofii, celkový přístup, jakým organizace přistupuje k testování a hlavní cíle testování
- Strategie testování, ta se konkrétněji zabývá způsoby a požadavky jakými je testování prováděno, zachovává však dostatečnou míru abstrakce, aby mohla být aplikována na další skupiny projektů
- Hlavní plán testování, plán je specificky zaměřen na konkrétní projekt, je v něm popsáno, jaká bude zvolena strategie testování v dané situaci
- Plán pro jednotlivé úrovně testování, jedná se o velice detailní plán využívaný u rozsáhlejších projektů, kromě hlavního plánu testování se vytvářejí plány i pro jednotlivé úrovně testování.

Při vytváření plánů testování mohou nastat situace, hlavně v ranných fázích projektu, kdy je nedostatek informací či jsou informace neúplné, v takových případech je nutné identifikovat tyto oblasti a zajistit jejich doplnění přiřazením očekávaného datumu úpravy dokumentu a přiřazení zodpovědnosti konkrétní osobě.

Plán testování musí být v souladu s projektovým plánem a lze ho tedy chápat jako aplikaci strategie a organizaci procesů testování, který slouží nejen manažerům, vývojářům, testovacímu týmu, pracovníkům zajišťujících kvalitu, ale může být přístupný i zákazníkům. Řada okolností a informací pro tvorbu testovacího plánu je zjištěna často až během procesu testování je proto důležité nekoncipovat plán příliš rigidně, ale ponechat prostor i pro nepředvídané situace. [18]

4 Vlastní práce

Praktická část diplomové práce bude řešit reálný projekt z oblasti inzertních portálů. Projekt je stále ve fázi vývoje nových funkčností, všechny implementované funkčnosti jsou po vývoji nasazeny na testovací prostředí a předány testovacímu týmu k ověření funkčnosti.

4.1 Popis testovaného systému a jeho funkcionalit

Inzertní portál je webová stránka, která slouží pro zveřejňování inzerátů na prodej nebo koupi nového či použitého zboží. Inzertní portál je úzce zaměřen na oblast sportovních dílů vozidel. Motivací pro vytvoření takového portálu byla snaha vlastníka umožnit uživatelům snadnější vyhledávání v inzerátech, které je podle jeho slov na současných inzertních portálech příliš komplikované. Další výhodou tohoto portálu je personalizovaný obsah a zobrazení inzerátů, které by se mohly uživateli líbit na základě jeho chování na portále. Pro vložení inzerátů a další pokročilé funkce portálu je tudíž nezbytná registrace za účelem vytvoření vlastního účtu. Přihlášený uživatel, který prošel procesem registrace může následně přidávat inzeráty a využívat další a jak již zmíněné výhody inzertního portálu.

Testování implementovaných funkcionalit probíhalo v ranné fázi celého projektu vývoje portálu. Proces registrace a přihlášení do webové aplikace byl iniciován uživatelem z jeho webového prohlížeče na stolním nebo mobilním zařízení.

Registrační proces je nezbytným krokem pro vytvoření vlastního účtu na inzertním portálu, na základě zadaných vstupů od uživatele provádí systém validaci vstupů a pokud jsou splněny všechny požadavky ukládá tyto informace do relační databáze.

Přihlášení do inzertního portálů je realizováno na základě získání autentizační informace od uživatele, kterou systém ověřuje oproti údajům ve své databázi. Autentizační informace uživatele je založena na znalosti kombinace emailu a hesla do inzertního portálu.

4.2 Plán testování

V této kapitole bude řešen návrh testovacího plánu, který tvoří strategii a organizuje procesy v testování nově implementovaných funkcionalit inzertního portálu zaměřeného na nákup a prodej sportovních dílů. Plán částečně vychází z dokumentace dostupné pro proces registrace a přihlášení uživatele.

4.2.1 Vymezení požadavků

Testování se zaměřuje na proces registrace a proces přihlášení uživatele do systému, tyto funkce jsou klíčové pro další vyvíjené funkcionality portálu. Z tohoto důvodu budou testovány základní a alternativní scénáře, které mohou nastat. Testování bude probíhat na grafickém uživatelském rozhraní portálu.

4.2.2 Cíle testování

Hlavním cílem testování je ověření, že implementované funkcionality splňují specifikované funkční požadavky definované v dokumentaci. Testování bude zaměřeno jak na funkční požadavky, tak i na nefunkční požadavky z pohledu logického návrhu funckionalit inzertního portálu.

Vedlejším cílem testování bude identifikace možných rizik a problémů, které mohou ohrozit časový harmonogram projektu a ohrozit tak celý proces testování.

4.2.3 Přístup k testování

Testování vychází z analýzy požadavků. Plánování, navrhování a odhadování testů je tvořeno na základě analýzy specifikačních požadavků aplikace. Mimo zmíněné, vychází testování částečně i ze znalostí a zkušeností testovacího týmu z podobných aplikací.

4.2.4 Role a odpovědnosti

Role	Odpovědnosti
Manažer testování	Řízení a zajišťování prostředků pro potřeby projektu, komunikace se členy týmu.
Analytik	Návrh řešení, tvorba a údržba technické dokumentace, komunikace se členy týmu.
Vývojář	Vývoj a implementace navrženého řešení a funkcionalit, opravy nalezených defektů, konfigurace prostředí, komunikace se členy týmu.
Tester	Vytvoření plánu testování, návrh testovacích případů, exekuce testovacích případů, zaznamenávání nalezených defektů, správa nalezených defektů, přetestování opravených defektů, příprava testovacích dat, komunikace se členy týmu.

Tabulka 1 - Role a odpovědnosti [vlastní zpracování]

4.2.5 Vstupní a výstupní kritéria

Vstupní kritéria

- Testovací prostředí je nakonfigurováno a připraveno k testování
- Je dostupná dokumentace potřebná k testování a k ověření správného chování aplikace
- Jsou dostupné veškeré potřebné nástroje pro testování
- Testovací scénáře jsou vytvořené a připravené pro exekuci

Výstupní kritéria

- Všechny testovací scénáře jsou provedeny
- Defekty s nejvyšším stupněm závažnosti a priority jsou vyřešeny
- Rizikové oblasti aplikace jsou otestovány

4.2.6 Pozastavení projektu a požadavky na jeho obnovení

Podmínky pro pozastavení projektu

- Aplikace obsahuje mnoho závažných defektů, které blokují další testování
- Je potřeba provést výraznou změnu v návrhu aplikace
- Přiřazené zdroje procesu testování nejsou k dispozici

Podmínky pro obnovení projektu

Obnovení projektu je možné pouze za předpokladu, že jsou vyřešeny všechny problémy, které zapříčinily jeho pozastavení.

4.2.7 Testovací strategie

Testovací tým se seznámí s aplikací a testovanými funkcionalitami z dostupných dokumentů. Na základě nastudovaných informací proběhne návrh a vytvoření testovacích scénářů.

Testovací scénáře budou vytvořeny technikou návrhu na základě dostupných případů užití testovaných funkcionalit, k nimž bude testovacím týmem vytvořena specifikace případů užití.

Testovací scénáře budou manuálně prováděny testovacím týmem. Každý otestovaný scénář bude ve jednom z uvedených stavů:

- Passed – test prošel
- Failed – test selhal
- N/A (Not Available) – test není možné dokončit

Výsledky testovacích scénářů budou zaznamenávány v dokumentu. Nalezené chyby budou testovacím týmem reportovány v aplikaci Trello. Opraveným chybám bude přiřazen příslušný stav a budou testovacím týmem přetestovány.

Testování bude probíhat na testovacím prostředí, po splnění všech cílů testování včetně opravy kritických defektů, bude aplikace nasazena na akceptační prostředí.

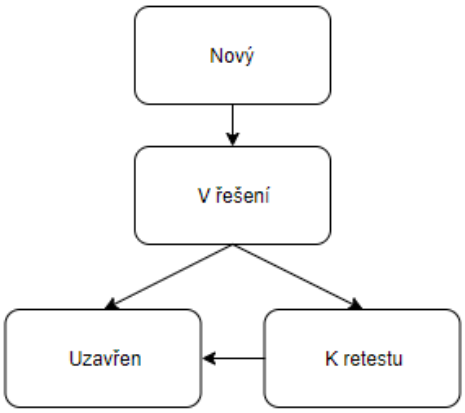
Typy testů

Na základě dostupných specifikací a dokumentací bude zvolen přístup gray-box testování. Funkční testy budou navrhovány z uživatelského pohledu a testovány přes grafické rozhraní aplikace. Na základě detailnější znalosti fungování aplikace budou vytvářeny testovací případy s cílem pokrýt i specifikované alternativní scénáře, které by nebyly dostatečně otestovány.

Dalším typem prováděných testů bude exploratorní testování.

Životní cyklus defektu

Nalezené defekty jsou reportovány v aplikaci Trello a vždy přiřazeny konkrétní osobě. Obrázek níže znázorňuje stavy, kterých může defekt nebývat.



Obrázek 9 - Životní cyklus defektu 2 [vlastní zpracování]

Závažnost a priorita defektů

Pro kategorizaci nalezených defektů je každému defektu testerem přiřazen stupeň závažnosti a priorita podle povahy chyby a dopadu na testování. Každý nalezený defekt musí mít přiřazený správný stupeň závažnosti a prioritu, za správné určení atributů je zodpovědné tester.

Stupeň	Závažnost	Popis
1	Kritická	Kritický defekt systému, daná funkcionalita nefunguje, neexistuje workaround, nedá se pokračovat.
2	Vysoká	Defekt ovlivňuje významné funkce systému, existuje workaround, systém je možné v omezené míře používat.
3	Střední	Defekt neovlivňuje významné funkce systému, systém je možné s obtížemi používat.
4	Nízká	Jedná se o drobné nedostatky typu kosmetických chyb.

Tabulka 2 - Závažnost defektů [vlastní zpracování]

Stupeň	Priorita	Popis
1	Kritická	Není možné pokračovat v testování nebo vývoji, blokační defekt, opravu je nutné provést okamžitě.
2	Vysoká	Částečně blokuje testování nebo vývoj, oprava je nutná ihned jak to bude možné.
3	Střední	Bez dopadu na proces testování, oprava je prováděna v rámci běžného pořadí opravy defektů.
4	Nízká	Oprava defektu není v současné chvíli důležitá, oprava přijde na řada po opravení důležitějších defektů.

Tabulka 3 - Priorita defektů [vlastní zpracování]

4.2.8 Identifikace rizik a problémů

Před začátkem testovacího procesu byla testovacím týmem identifikována možná rizika, která mohou ohrozit projekt:

- Nedostatečná specifikace, neaktuální dokumentace
- Chyby v návrhu funkcionalit
- Problémy při nasazování a konfiguraci testovacího prostředí
- Nedodržení termínů oprav defektů a zdržení testování

4.2.9 Časový harmonogram

Časový harmonogram	
Tvorba technické dokumentace	2.1.2018 – 8.1.2018
Implementace	9.1.2018 – 19.1.2018
Test analýza	22.1.2018 – 31.1.2018
Nasazení na testovací prostředí	1.2.2018 – 2.2.2018
Testování aplikace	5.2.2018 – 23.2.2018
Nasazení na akceptační prostředí	26.2.2018-27.2.2018

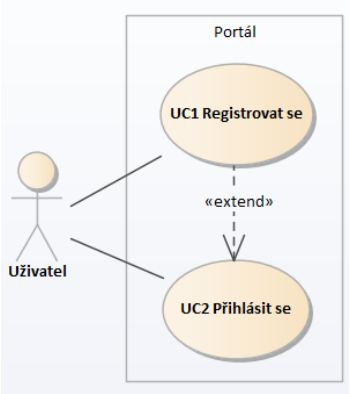
Tabulka 4 - Časový harmonogram [vlastní zpracování]

4.2.10 Definice názvosloví

- Defekt – nesoulad s technickou dokumentací, funkčními požadavky nebo jiná chyba nalezená během procesu testování

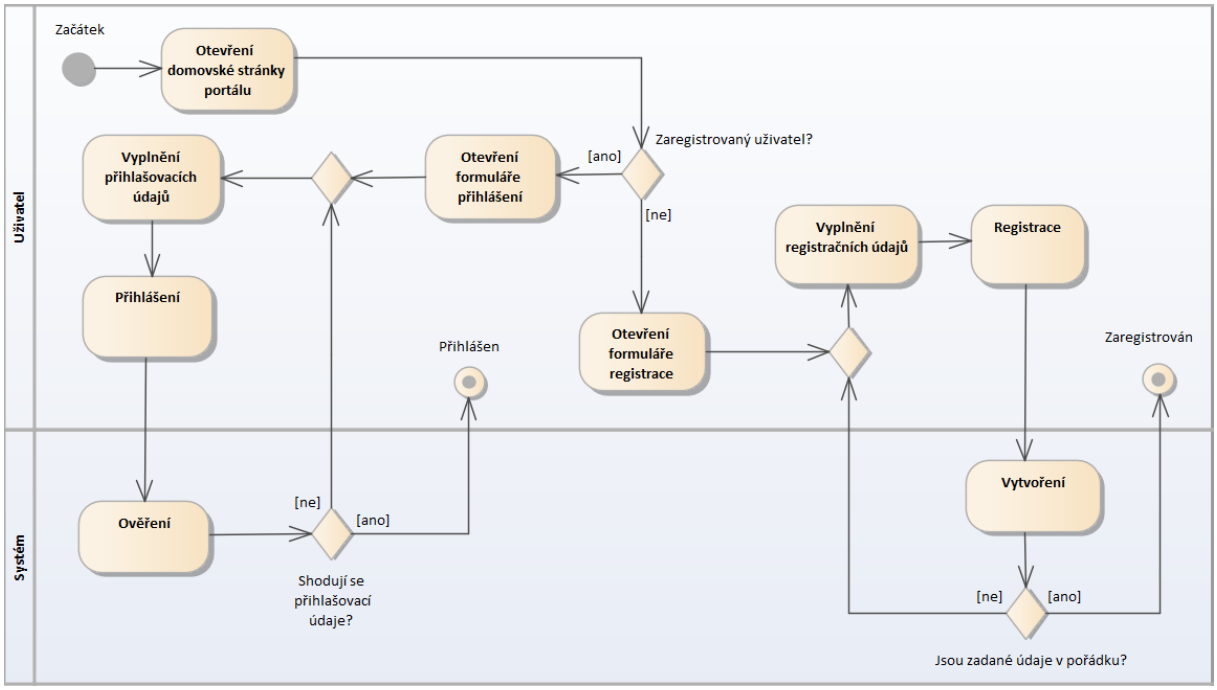
4.3 Dokumentace k případům užití

Obrázek č. 10 znázorňuje chování systému z pohledu uživatele, účelem diagramu je popsání testovaných funkcionalit, které se od systému očekávají. Vnitřní logika systému není zachycena, proto digram popisuje pouze to, co má systém umožňovat, ale už není definováno, jak toho bude docíleno.



Obrázek 10 - Diagram případu užití [vlastní zpracování]

Diagram aktivit znázorněný na dalším obrázku již popisuje konkrétní sled aktivit vedoucích k určitému cíli. Z dokumentace dostupný diagram reprezentuje obě testované funkčnosti inzertního portálu.



Obrázek 11 - Diagram aktivit [vlastní zpracování]

Validace a zprávy

Součástí dostupné dokumentace je mimo dostupné diagramy i soupis validačních pravidel, které systém provádí z důvodu zajištění konzistentnosti databáze a kvůli zamezení uložení nekorektních dat.

Atribut	Pravidlo
Celé jméno	Alespoň 2 znaky
Email	Emailová adresa ve tvaru „vzor@vzor.cz“
Telefon	Alespoň 9 numerických znaků s možností znaménka „+“ v předvolbě
Heslo	Alespoň 8 znaků

Tabulka 5 - Validační pravidla vstupních hodnot [vlastní zpracování]

Tabulka č. 6 obsahuje číselník informačních a chybových zpráv zobrazovaných uživateli systémem.

Identifikátor	Text
MSG001	Zadejte prosím vaše jméno.
MSG002	Zadejte prosím alespoň dva znaky.
MSG003	Zadejte prosím váš e-mail.
MSG004	Zadejte prosím správný formát e-mailu.
MSG005	Zadejte prosím váš telefon.
MSG006	Telefonní číslo není ve správném tvaru.
MSG007	Zadejte prosím heslo.
MSG008	Heslo musí mít alespoň 8 znaků.
MSG009	Uživatel byl úspěšně registrován.
MSG010	Uživatel s tímto e-mailem již existuje.
MSG011	Přihlášení se nezdařilo. Pokud ještě nemáte účet registrujte se zde. Pokud jste zapomněli heslo můžete ho obnovit zde.

Tabulka 6 - Číselník systémových zpráv [vlastní zpracování]

Dostupná funkční specifikace tvoří předpoklad pro vytvoření specifikace případů užití, která je jedním z cílů testovacího týmu. Takto specifikované případy užití tvoří ideální případ pro pokrytí základních a alternativních scénářů, které mohou nastat u obou funkcionalit.

4.4 Specifikace případů užití

Dle strategie testování definované v plánu testování, budou testovací scénáře navrženy technikou návrhu podle případů užití. Testována bude funkcionalita „UC1 Registrovat se“ a „UC2 Přihlásit se“. Dostupnou dokumentací, ze které bude návrh testovacích případů vycházet je diagram případu užití a diagram aktivit, který znázorňuje interakci uživatele se systémem.

4.4.1 UC1 Registrovat se

Cíl případu užití

Případ užití „*UC001_Registrace_uzivatele“* umožňuje uživateli na základě zadaných vstupů registraci do systému.

Aktéři

- Uživatel
- Systém

Vstupní podmínky

- Uživatel se musí nacházet na formuláři registrace.

Výstupní podmínky

- Uživateli je vytvořen účet pro přihlášení do systému.

Základní scénář

1. Systém zobrazí uživateli formulář pro vyplnění celého jména, emailu, telefonního čísla a hesla
2. Uživatel vyplní celé jméno, email, telefonní číslo a heslo
3. Systém zvaliduje zadané vstupy
4. Uživatel odešle vyplněný formulář
5. Systém zvaliduje email a zobrazí MSG009

Alternativní scénář 1

3.1 – Uživatel zadá neplatný vstup, systém zobrazí příslušnou chybovou hlášku a nedovolí uživateli formulář odeslat.

3.2 – Uživatel opraví neplatný vstup, scénář pokračuje 2. krokem základního scénáře.

Alternativní scénář 2

5.1 – V systému existuje uživatel se stejným emailem, systém zobrazí MSG010.

5.2 – Uživatel si zvolí jiný email, scénář pokračuje 2. krokem základního scénáře.

4.4.2 UC2 Přihlásit se

Cíl případu užití

Případ užití „*UC002_Prihlaseni_uzivatele“* umožňuje uživateli na základě zadaných vstupů přihlášení do systému.

Aktéři

- Uživatel
- Systém

Vstupní podmínky

- Uživatel se musí nacházet na formuláři přihlášení.
- Uživatel musí mít dokončený proces registrace (UC1 Registrovat se)

Výstupní podmínky

- Uživatel se přihlásí do systému.

Základní scénář

1. Systém zobrazí uživateli formulář pro vyplnění emailu a hesla
2. Uživatel vyplní email a heslo
3. Systém zvaliduje zadané vstupy
4. Uživatel odešle vyplněný formulář
5. Systém autentizuje uživatele

Alternativní scénář 1

3.1 – Uživatel zadá neplatný vstup, systém zobrazí příslušnou chybovou hlášku a nedovolí uživateli formulář odeslat.

3.2 – Uživatel opraví neplatný vstup, scénář pokračuje 2. krokem základního scénáře.

Alternativní scénář 2

5.1 – Uživatel zadá neplatné heslo k existujícímu účtu, systém zobrazí MSG011.

5.2 – Uživatel zadá platné heslo, scénář pokračuje 2. krokem základního scénáře.

4.5 Test analýza

Tato kapitola se bude věnovat samotnému návrhu testovacích scénářů. Na základě vytvořených specifikací má testovací tým dostatečné podklady pro tvorbu konkrétních testů. Testy jsou navrženy s cílem pokrýt specifikované základní a alternativní scénáře.

Paralelně s prováděním funkčních testů aplikace bude probíhat fáze exploratorních testů. Pro tyto testy nebudou vytvářeny testovací scénáře, nalezené defekty však budou reportovány stejně jako u funkčních testů. Exploratorní testy budou založeny na předpokladu, že má testovací tým dostatečné znalosti z testování obdobných projektů.

4.5.1 Návrh testovacích případů

Název testu	TC001 Registrace – základní scénář	**Priorita**	1
Typ testu	FAT	**Datum provedení testu**	
Předpoklady	Email není v systému registrován	**Tester**	JL
Krok	**Popis**	**Očekávané chování**	**Výsledek**
1	Otevři formulář pro registraci na úvodní stránce	Systém zobrazí registrační formulář	
2	Korektně vyplň celé jméno, email, telefonní číslo a heslo	Systém zvaliduje zadané vstupy a nezobrazí žádnou chybovou hlášku	
3	Odešli vyplněný formulář	Systém zvaliduje email a zobrazí MSG009	

Tabulka 7 - Testovací případ TC001 [vlastní zpracování]

Název testu	TC002 Registrace – validace celé jméno	**Priorita**	2
Typ testu	FAT	**Datum provedení testu**	
Předpoklady	Email není v systému registrován	**Tester**	JL
Krok	**Popis**	**Očekávané chování**	**Výsledek**
1	Otevři formulář pro registraci na úvodní stránce	Systém zobrazí registrační formulář	
2	Korektně vyplň email, telefonní číslo a heslo Pole pro celé jméno nech prázdné	Systém zvaliduje zadané vstupy a zobrazí MSG001	
3	Do pole pro celé jméno doplň 1 znak	Systém zvaliduje zadané vstupy a zobrazí MSG002	
4	Korektně vyplň celé jméno a odešli formulář	Systém zvaliduje email a zobrazí MSG009	

Tabulka 8 - Testovací případ TC002 [vlastní zpracování]

Název testu	TC003 Registrace – validace email	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	Email není v systému registrován	Tester	JL
Krok	**Popis**	**Očekávané chování**	**Výsledek**
1	Otevři formulář pro registraci na úvodní stránce	Systém zobrazí registrační formulář	
2	Korektně vyplň celé jméno, telefonní číslo a heslo Pole pro email nech prázdné	Systém zvaliduje zadané vstupy a zobrazí MSG003	
3	Do pole pro email vyplň email v nesprávném tvaru	Systém zvaliduje zadané vstupy a zobrazí MSG004	
4	Korektně vyplň email a odešli formulář	Systém zvaliduje email a zobrazí MSG009	

Tabulka 9 - Testovací případ TC003 [vlastní zpracování]

Název testu	TC004 Registrace – validace telefonní číslo	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	Email není v systému registrován	Tester	JL
Krok	**Popis**	**Očekávané chování**	**Výsledek**
1	Otevři formulář pro registraci na úvodní stránce	Systém zobrazí registrační formulář	
2	Korektně vyplň celé jméno, email a heslo Pole pro telefonní číslo nech prázdné	Systém zvaliduje zadané vstupy a zobrazí MSG005	
3	Do pole pro telefonní číslo vyplň telefonní číslo v nesprávném tvaru	Systém zvaliduje zadané vstupy a zobrazí MSG006	
4	Korektně vyplň telefonní číslo a odešli formulář	Systém zvaliduje email a zobrazí MSG009	

Tabulka 10 - Testovací případ TC004 [vlastní zpracování]

Název testu	TC005 Registrace – validace heslo	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	Email není v systému registrován	Tester	JL
Krok	**Popis**	**Očekávané chování**	**Výsledek**
1	Otevři formulář pro registraci na úvodní stránce	Systém zobrazí registrační formulář	
2	Korektně vyplň celé jméno, email a telefonní číslo Pole pro heslo nech prázdné	Systém zvaliduje zadané vstupy a zobrazí MSG007	
3	Do pole pro heslo vyplň heslo kratší než 8 znaků	Systém zvaliduje zadané vstupy a zobrazí MSG008	
4	Korektně vyplň heslo a odešli formulář	Systém zvaliduje email a zobrazí MSG009	

Tabulka 11 - Testovací případ TC005 [vlastní zpracování]

Název testu	TC006 Registrace – validace existující email	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	V systému existuje zaregistrovaný email	Tester	JL
Krok	**Popis**	**Očekávané chování**	**Výsledek**
1	Otevři formulář pro registraci na úvodní stránce	Systém zobrazí registrační formulář	
2	Korektně vyplň celé jméno, telefonní číslo a heslo Do pole pro email vyplň v systému existující email	Systém zvaliduje zadané vstupy a nezobrazí žádnou chybovou hlášku	
3	Odešli vyplněný formulář	Systém zvaliduje email a zobrazí MSG010	
4	Korektně vyplň email a odešli formulář	Systém zvaliduje email a zobrazí MSG009	

Tabulka 12 - Testovací případ TC006 [vlastní zpracování]

Název testu	TC007 Přihlášení – základní scénář	Priorita	1
Typ testu	FAT	Datum provedení testu	
Předpoklady	Účet s dokončeným procesem registrace	Tester	JL
Krok	Popis	Očekávané chování	Výsledek
1	Otevři formulář pro přihlášení na úvodní stránce	Systém zobrazí přihlašovací formulář	
2	Korektně vyplň email a heslo	Systém zvaliduje zadané vstupy a nezobrazí žádnou chybovou hlášku	
3	Odešli vyplněný formulář	Systém autentizuje a přihlásí uživatele	

Tabulka 13 - Testovací případ TC007 [vlastní zpracování]

Název testu	TC008 Přihlášení – validace email	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	Účet s dokončeným procesem registrace	Tester	JL
Krok	Popis	Očekávané chování	Výsledek
1	Otevři formulář pro přihlášení na úvodní stránce	Systém zobrazí přihlašovací formulář	
2	Korektně vyplň heslo Pole pro email nech prázdné	Systém zvaliduje zadané vstupy a zobrazí MSG003	
3	Do pole pro email vyplň email v nesprávném tvaru	Systém zvaliduje zadané vstupy a zobrazí MSG004	
4	Korektně vyplň email a odešli formulář	Systém autentizuje a přihlásí uživatele	

Tabulka 14 - Testovací případ TC008 [vlastní zpracování]

Název testu	TC009 Přihlášení – validace heslo	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	Účet s dokončeným procesem registrace	Tester	JL
Krok	Popis	Očekávané chování	Výsledek
1	Otevři formulář pro přihlášení na úvodní stránce	Systém zobrazí přihlašovací formulář	
2	Korektně vyplň email Pole pro heslo nech prázdné	Systém zvaliduje zadané vstupy a zobrazí MSG007	
3	Do pole pro heslo vyplň heslo kratší než 8 znaků	Systém zvaliduje zadané vstupy a zobrazí MSG008	
4	Korektně vyplň heslo a odešli formulář	Systém autentizuje a přihlásí uživatele	

Tabulka 15 - Testovací případ TC009 [vlastní zpracování]

Název testu	TC010 Přihlášení – nesprávné heslo	Priorita	2
Typ testu	FAT	Datum provedení testu	
Předpoklady	Účet s dokončeným procesem registrace	Tester	JL
Krok	Popis	Očekávané chování	Výsledek
1	Otevři formulář pro přihlášení na úvodní stránce	Systém zobrazí přihlašovací formulář	
2	Korektně vyplň v systému existující email Do pole pro heslo vyplň nesprávné heslo ve validním tvaru k existujícímu emailu	Systém zvaliduje zadané vstupy a nezobrazí žádnou chybovou hlášku	
3	Odešli vyplněný formulář	Systém zvaliduje kombinaci emailu a hesla a zobrazí MSG011	
4	Korektně vyplň email a heslo zaregistrovaného uživatele a odešli formulář	Systém autentizuje a přihlásí uživatele	

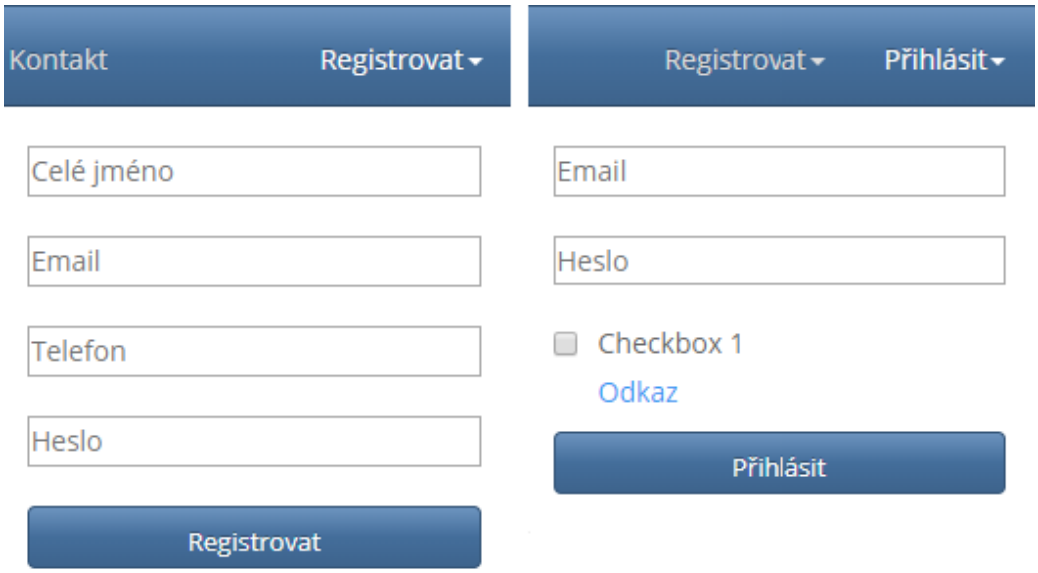
Tabulka 16 - Testovací případ TC010 [vlastní zpracování]

4.6 Exekuce testovacích případů

Kapitola 4.6 Exekuce testovacích případů se bude zabývat samotnou realizací navržených testů. Funkční testy budou prováděny členy testovacího týmu podle definované priority. Po provedení testu bude testovacímu scénáři přiřazen odpovídající stav podle výsledku. V případě nalezení defektu, bude navíc označen příslušným stavem i krok ve kterém byl defekt nalezen.

Defekty nalezené na základě provedených funkčních testů budou reportovány v aplikaci Trello.

Na následujícím obrázku jsou znázorněny oba formuláře, na kterých bude probíhat testování.



Obrázek 12 - Registrační a přihlašovací formulář [vlastní zpracování]

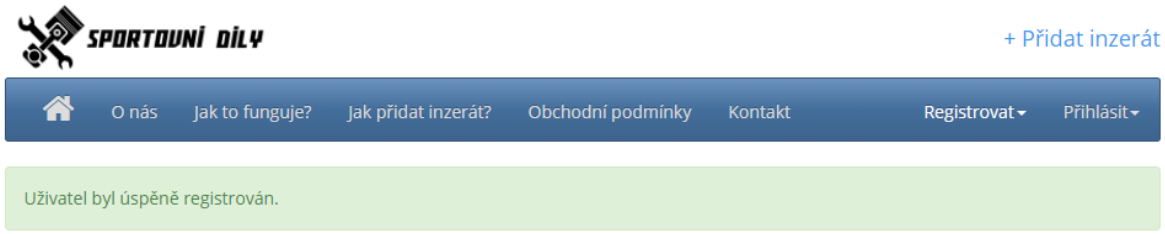
Exekuce testovacího scénáře TC001

Testovací scénář testuje základní průchod procesem registrace, je to situace kdy v procesu nenastane žádná alternativa.

První krok scénáře ověřuje samotné zobrazení formuláře. Ve druhém kroku jsou uživatelem validně vyplněny vstupy v podobě celého jména, emailu, telefonního čísla a hesla. Systém po každém opuštění pole formuláře provádí validaci. Po vyplnění všech polí validními údaji je formulář uživatelem odeslán, po odeslání vyplněného formuláře

probíhá validace emailové adresy. Po úspěšné validaci je uživateli zobrazena informační hláška a proces registrace končí.

Na obrázku č. 13 je zachycena obrazovka aplikace po dokončení testovacího scénáře. Očekávané chování systému bylo ve všech krocích scénáře naplněno. Po odeslání formuláře byla uživateli zobrazena informační hláška, test lze označit stavem *„Passed“*.



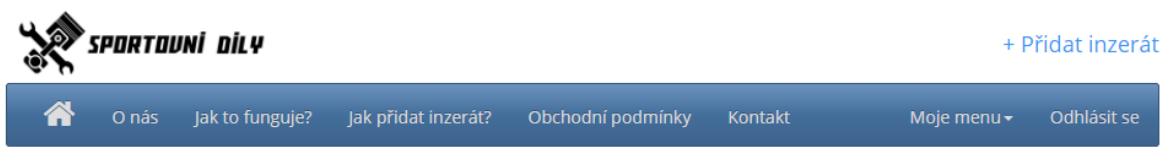
Obrázek 13 - Úspěšné dokončení registrace [vlastní zpracování]

Exekuce testovacího scénáře TC007

Testovací scénář je také scénář s prioritou 1, je tedy proveden jako další v pořadí. Tento scénář rovněž ověřuje základní průchod, v tomto případě procesu přihlášení. Prerekvizitou pro vykonání tohoto scénáře je účet s dokončeným procesem registrace. Pro přihlášení lze použít přihlašovací údaje vytvořené v předchozím testu.

První krok ověřuje zobrazení formuláře, v dalším kroku je uživatelem vyplněn email s příslušným heslem. Po opuštění pole systém provádí validaci správného tvaru obou hodnot. V posledním kroku je formulář uživatelem odeslán, systém provede autentizaci a uživatele přihlásí.

Stav systému po provedení testu je zachycen na obrázku níže. Uživateli jsou zpřístupněny další funkce aplikace a je mu umožněno odhlášení. Stejně jako první testovací případ, lze považovat tento test se všemi jeho kroky za splněný a označit ho stavem *„Passed“*.



Obrázek 14 - Úspěšné přihlášení uživatele [vlastní zpracování]

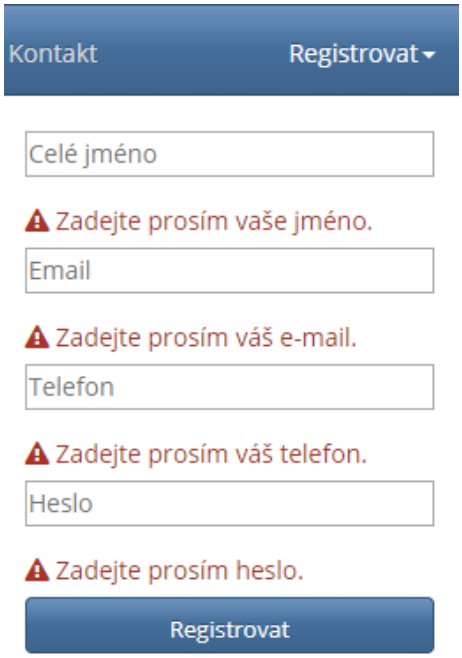
Exekuce testovacích scénářů TC002, TC003, TC004, TC005

Cílem testovacích scénářů je ověření alternativního způsobu dokončení procesu registrace, pokud uživatel vyplní do polí pro celé jméno, email, telefon nebo heslo nevalidní vstup. Testy jsou navíc zaměřeny na ověření zobrazení chybových hlášek upozorňujících na nevalidní vstupy. Proces registrace je možné dokončit za předpokladu, že jsou ve všech polích vyplněny vstupy, které neporušují validační pravidla konkrétních polí.

V prvním kroku je ověřeno zobrazení registračního formuláře, v následujících krocích 2 a 3 jsou uživateli zobrazeny po vyplnění vstupů chybové hlášky. První chybová hláška je uživateli zobrazena, pokud nechá pole prázdné, zobrazení druhé chybové hlášky, která upozorňuje na nevalidní vstup je uživateli zobrazena, pokud zadaný vstup neodpovídá validačním pravidlům konkrétních polí.

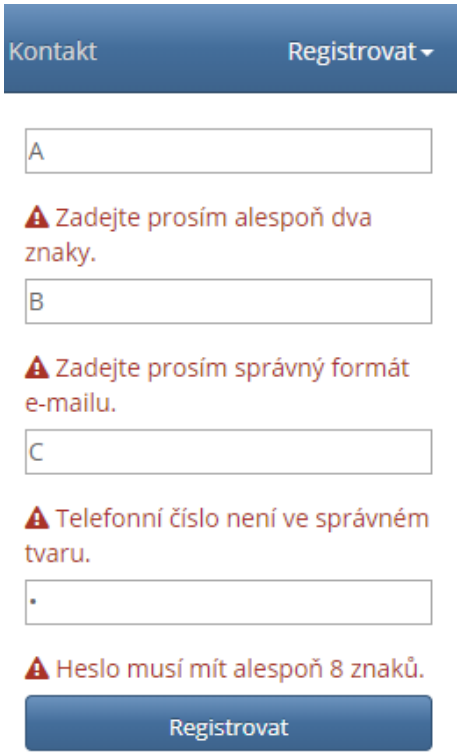
Poslední 4. krok každého testovacího scénáře ověřuje možnost dokončení procesu registrace.

Na obrázku č. 15 jsou zobrazeny chybové hlášky upozorňující uživatele na povinnost vyplnění všech polí registračního formuláře.



Obrázek 15 - Chybové zprávy registračního formuláře 1 [vlastní zpracování]

Obrázek č. 16 zachycuje konkrétní chybové hlášky zobrazované, pokud zadaný vstup konkrétního pole porušuje jeho specifická validační pravidla.



Obrázek 16 - Chybové zprávy registračního formuláře 2 [vlastní zpracování]

Po vyplnění všech požadovaných polí validními vstupy a odeslání registračního formuláře bylo v posledním kroku všech testovaných scénářů dosaženo stejného stavu systému jako při exekuci testu TC001. Všechny provedené testy je tedy možné označit stavem *„Passed“*.

Exekuce testovacího scénáře TC006

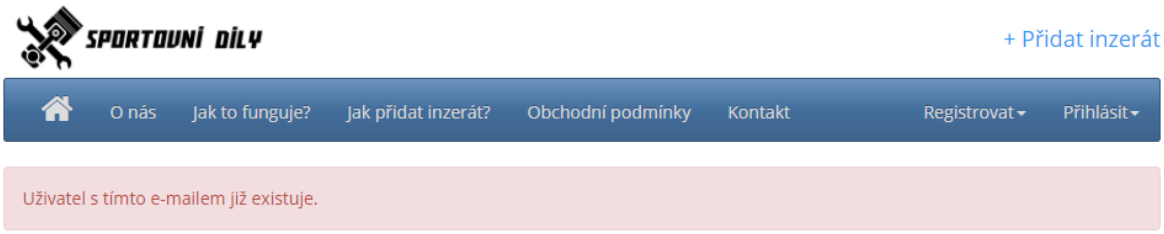
Předpokladem pro provedení testovacího scénáře jsou již vytvořená testovací data v podobě vytvořených uživatelských účtů s dokončeným procesem registrace. Cílem testu je ověření nemožnosti vytvoření uživatelského účtu s emailovou adresou, která je již v systému zaregistrovaná. Tato skutečnost je uživateli sdělena zobrazením chybové hlášky.

První krok testovacího scénáře ověřuje dostupnost registračního formuláře, ve druhém kroku jsou uživatelem vyplněna všechna povinná pole validními vstupy. Vstupy jsou po každém opuštění pole validovány systémem, emailová adresa se v tomto kroku validuje pouze z pohledu správného formátu. Systém nezobrazuje žádnou chybovou hlášku jeli vyplněna v systému již existující adresa.

Odesláním vyplněného formuláře ve 3. kroku testovacího scénáře probíhá validace emailové adresy vůči registrovaným emailovým adresám v databázovém systému. Na tuto skutečnost je uživatel upozorněn systémem zobrazenou chybovou hláškou.

Poslední krok scénáře končí úspěšnou registrací uživatele po vyplnění emailové adresy, která není v systému zaregistrovaná.

Na následujícím obrázku je zachyceno očekávané chování po odeslání registračního formuláře s existující emailovou adresou v systému.



Obrázek 17 - Chybová zpráva při registraci s existujícím emailem [vlastní zpracování]

Testovacím případem bylo ověřeno očekávaní chování systému při zadaní existující emailové adresy v procesu registrace. Po opravení vstupu bylo uživateli umožněno dokončit proces registrace. Test je možné označit stavem *„Passed“*.

Exekuce testovacích scénářů TC008, TC009

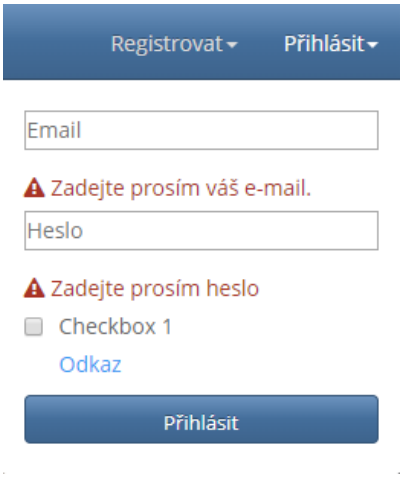
Smyslem těchto testů je ověření prvního alternativního scénáře procesu přihlášení. Uživateli není umožněno odeslat formulář, pokud obsahuje nevalidní vstupy, na tuto skutečnost je uživatel upozorněn zobrazenými chybovými hláškami. Formulář lze odeslat pouze s validně vyplněnými poli.

První krok scénáře ověřuje dostupnost přihlašovacího formuláře na úvodní stránce inzertního portálu. Po vyplnění nevalidních vstupů ve 2. a 3. kroku je uživatel na

skutečnost upozorněn příslušnou chybovou hláškou. Druhý krok obou scénářů ověřuje zobrazení chybové hlášky po nevyplnění polí, ve třetím kroku je testováno zobrazení chybové hlášky, pokud pole nesplňují specifikovaná validační pravidla.

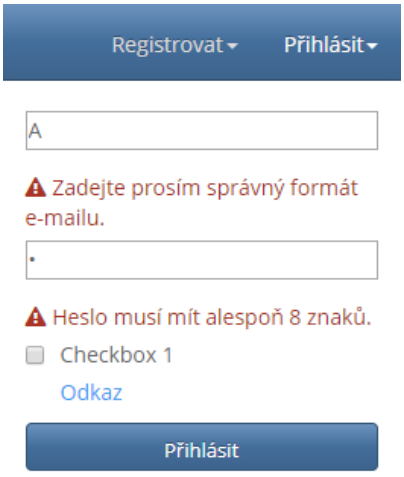
V posledním kroku je ověřeno přihlášení uživatele po opravě nevalidních vstupů a zadání korektních přihlašovacích údajů.

Očekávané chování systému po splnění 2. kroku je souhrnně zobrazeno na obrázku č. 18 níže.



Obrázek 18 - Chybové zprávy přihlašovacího formuláře 1 [vlastní zpracování]

Na obrázku č. 19 je zobrazeny chybové hlášky informující uživatele na nevalidně vyplněné vstupy na přihlašovacím formuláři.



Obrázek 19 - Chybové zprávy přihlašovacího formuláře 2 [vlastní zpracování]

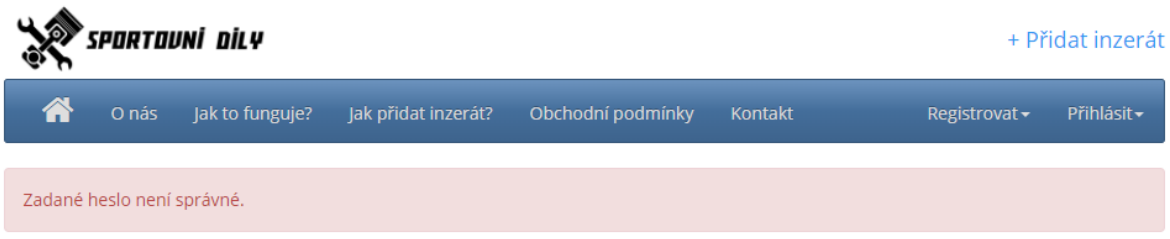
Očekávané chování bylo splněno ve všech krocích testovacích scénářů, proto lze testy označit stavem *„Passed“*.

Exekuce testovacího scénáře TC010

Testovací scénář ověřuje očekávané chování systému při pokusu o přihlášení uživatele nesprávným heslem. Uživatel je o této skutečnosti informován na základě zobrazení příslušné chybové hlášky. Po zadání správného hesla je uživateli umožněno přihlášení ke svému účtu.

V prvním kroku scénáře je ověřena dostupnost přihlašovacího formuláře a vstupních polí. Ve druhém kroku zadává uživatel existující emailovou adresu, do pole pro heslo je vyplněno heslo, které neodpovídá existujícímu účtu. Heslo je vyplněno ve správném tvaru, tedy podle validačních pravidel tzn. řetězec s délkou alespoň 8 znaků. Autentizace uživatelského účtu na základě zadané emailové adresy a hesla probíhá po odeslání formuláře ve 3. kroku testovacího případu. Systém v tomto kroku reaguje zobrazením chybové hlášky. V posledním kroku je uživatel po vyplnění odpovídajícího hesla ke svému uživatelskému účtu přihlášen.

Chování systému ve 3. kroku nesplnilo očekávání, uživateli je zobrazena jiná chybová hláška než uvedená ve specifikaci. Zmíněný krok a celý testovacího scénáře je označen stavem *„Failed“*. Zobrazení jiné, než v dokumentaci specifikované hlášky je zobrazeno na obrázku č. 20 níže.



Obrázek 20 - Nesprávná chybová zpráva přihlašovacího formuláře [vlastní zpracování]

Exploratorní testování

Paralelně s prováděním funkčních testů probíhalo exploratorní testování, na základě provedených testů byla identifikována rizika v nedostatečných validačních pravidlech registračního formuláře. Validační pravidla byla příliš obecná, čímž docházelo k ukládání nesmyslných hodnot do databáze. Takto neošetřené vstupy snižovali bezpečnost aplikace.

Po opuštění polí registračního formuláře nebyla uživateli zobrazena, kvůli příliš obecným validačním pravidlům žádná chybová hláška. Takto vyplněný formulář prošel validací a po odeslání byl vytvořen účet.

Obrázek 21 - Příliš obecné validace polí registračního formuláře [vlastní zpracování]

Na základě defektu nalezeného funkčním testováním bylo exploratorní testování částečně zaměřeno na testování aplikace z pohledu bezpečnosti. Chování aplikace, popsané v nalezeném defektu z funkčních testů umožňovalo potenciálnímu útočníkovi získat přehled o zaregistrovaných účtech, opatření proti tomuto nedostatku nalezeném na přihlašovacím formuláři bylo sepsáno a reportováno přímo analytickému týmu.

Návrhy opatření a další nedostatky nalezené exploratorním testováním jsou blíže popsány v kapitole 5 Výsledky a diskuze.

4.7 Zaznamenání chyb

Defekty nalezené po provedení všech navržených testovacích scénářů budou v souladu s plánem testovaní reportovány v aplikaci Trello. Aplikace slouží jako jednoduchá elektronická tabule, princip reportování chyb je založen na vytvoření nalezeného defektu do příslušného sloupce elektronické tabule. Stav defektu je potom definován podle příslušného sloupce, ve kterém se defekt nachází.

Defekty nalezené exploratorním testováním byly sepsány a reportovány přímo analytickému týmu.

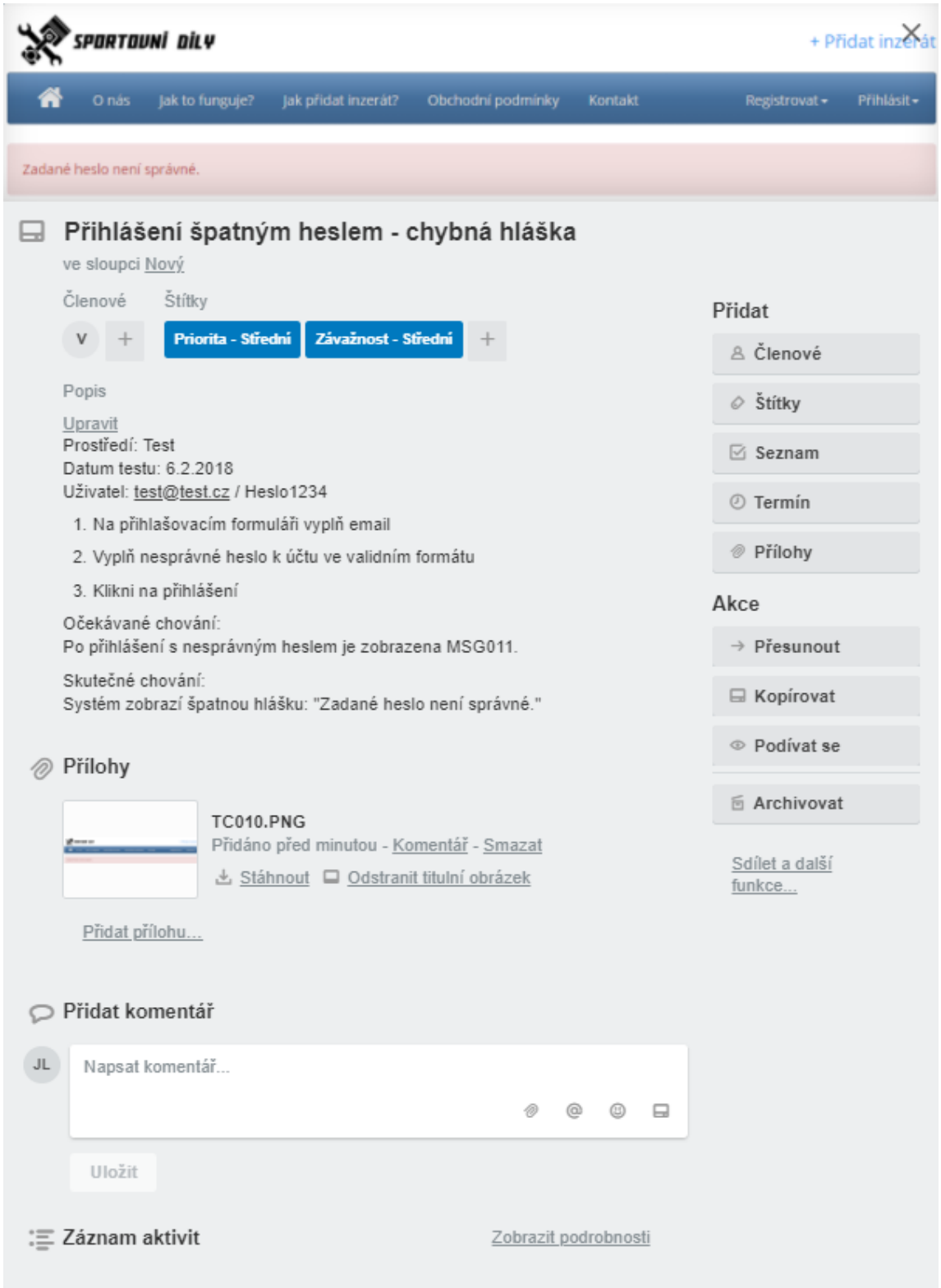
Exekucí testovacího scénáře TC010 byl odhalen defekt v podobě nesouhlasící chybové hlášky, která je uživateli zobrazena, pokud se přihlašuje nesprávným heslem. Vzniklou skutečnost je potřeba reportovat vývojovému týmu. Na základě povahy defektu je mu přiřazena příslušná závažnost a priorita.

Nalezený defekt nemá zásadní vliv na funkčnost systému ani dané funkcionality, neblokuje nijak proces dalšího testování ani jiné testovací scénáře. Střední závažnost byla defektu přiřazena, jelikož se jedná o chybu bezpečnostního charakteru. Defekt bude opraven ve standardním pořadí, tudíž mu byla přiřazena střední priorita.

Zaznamenaný defekt je doplněn o informace o čase a prostředí na kterém byl nalezen. Popis defektu obsahuje postup k nasimulování s testovacími daty. Snímek obrazovky se zobrazenou chybnou zprávou je přiložen v příloze defektu.

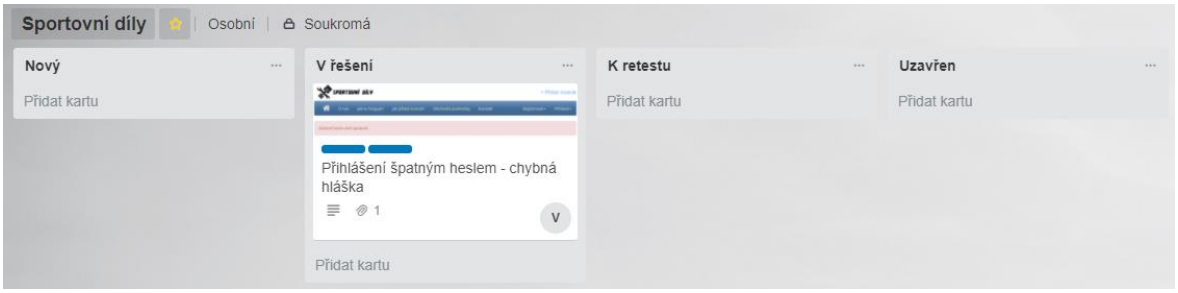
Defekt se nachází ve stavu *„Nový“* a je přiřazen vývojáři testované funkcionality.

Kompletní popis reportované chyby v aplikaci Trello je znázorněn na obrázku č. 22 níže:



Obrázek 22 - Vytvoření defektu v aplikaci Trello [vlastní zpracování]

Pozicí defektu v příslušném sloupci je určen jeho stav, na obrázku č. 23 je znázorněna situace, kdy je defekt přiřazen vývojáři, který se věnuje jeho opravě.



Obrázek 23 - Aktuální stav defektu [vlastní zpracování]

5 Zhodnocení výsledků

Úvod praktické části se věnuje popisu inzertního portálu a jeho testovaných funkcionalit, z požadavků vyplynula potřeba otestovat dvě nové funkcionality, které jsou klíčové pro další vývoj aplikace. Myšlenkou vlastníka je vytvoření inzertního portálu s personalizovaným obsahem na základě jeho chování, předpokladem proto bylo zajistit funkčnost a otestovat proces registrace a přihlášení do portálu.

Popis testovaného produktu byl výchozím bodem pro sestavení plánu testování, ve kterém byl specifikován zvolený postup prováděných testů a ověření aplikace. Testování aplikace vychází z analýzy požadavků, na základě, kterých bylo plánováno testování.

Role a odpovědnosti jednotlivých členů projektového týmu byly explicitně definovány v úvodu plánu testování. Rozhodnutí o začátku a konci projektu bylo provedeno na základě vstupních a výstupních kritérií. Vstupními kritérii pro započatí projektu bylo především dostupné testovací prostředí s implementovanými funkcionalitami z vývojového prostředí. Dalším předpokladem, bez kterého nemohl projekt začít byla vytvořená technická dokumentace testovaných částí projektu, před samotným procesem testování aplikace bylo nutné disponovat již vytvořenými funkčními testy.

Před samotným návrhem testovacích případů bylo nutností nastudování dokumentace projektu, podle úrovně dokumentace pak bylo rozhodnuto o návrhu testovacích případů na základě dostupných případů užití. Důležitou částí dokumentace, ze které návrh testů a především pak návrh specifikace případů užití vycházel, byl diagram aktivit obou testovaných procesů.

Na základě dokumentace byl rovněž zvolen typ testů. Funkční testy byly navrženy z uživatelského pohledu jen s minimální znalostí fungování vnitřních procesů aplikace. Paralelně s těmito testy probíhala analýza rizikových částí aplikace, ty pak byly testovány pomocí exploratorních testů, které vycházeli ze zkušeností testovacího týmu z podobných projektů.

Povinností testovacího týmu bylo na základě výše uvedených poznatků sestavení specifikace testovacích případů, který tvořily předpoklad pro vytvoření testů. Pro oba procesy registrace a přihlášení byly vytvořeny základní scénáře s možnými alternativními průběhy. Podle specifikací byly potom navrženy konkrétní testy.

Exekuce testovacích scénářů probíhala podle stanovené priority při návrhu testů. Testy s nejvyšší prioritou, které byly vykonány jako první a nebyla u nich nalezena žádná chyba, která by blokovala další průběh testů byl testy základního scénáře funkcionalit, tedy testy bez alternativního průběhu.

Ostatní testovací případy měly stejnou prioritu, nebylo tedy nutné určovat jejich pořadí podle kterého byly prováděny.

Na základě provedení všech navržených testovacích případů byla nalezena chyba ve zobrazení špatné chybové zprávy, která informuje uživatele o pokusu přihlášení s neplatným heslem. Chyba byla dle pravidel definovaných v plánu testování reportována v aplikaci Trello, byla určena její závažnost a priorita a byla přiřazena konkrétnímu vývojáři. Popis chyby tvořil především postup k nasimulování pro její reprodukci.

Exploratorním testováním bylo zjištěno, že je nalezená chyba bezpečnostního charakteru. Na základě konkrétní chybové hlášky, která se zobrazila uživateli, pokud zadal špatné heslo k existujícímu účtu, mohl útočník postupným zkoušením různých emailových adres a hesel zjistit skutečné emailové adresy zaregistrované v systému. Uživatelské účty nebyly chráněny možností zablokování přístupu po několika nesprávných pokusech o přihlášení, což by mohlo vést k tomu, že by útočník využil nějakou metodu k prolomení hesla. Jednoduchá ochrana proti takovému útoku by mohla být blokace účtu po několika špatných pokusech. O této skutečnosti by byl uživatel informován emailem, který by obsahoval odkaz pro odblokování. Další možností by mohlo být postupné navyšování času pro další pokus o přihlášení, pokud by bylo zadáno špatné heslo.

Další exploratorní testy byly díky objevení této zranitelnosti zaměřeny na bezpečnost aplikace. Potenciální zranitelnost, která byla objevena na registračním formuláři, byly příliš obecné validace polí, bez definování maximální délky řetězce. Pole pro zadání celého jména nebylo omezeno na zadání pouze alfabetických znaků, tudíž by útočník mohl využít další zranitelnosti v podobě SQL Injection nebo Cross-site scripting.

Zjištěné nedostatky nebyly reportovány v aplikaci Trello, byly předány k podrobnější analýze přímo analytickému týmu.

Projekt byl rozdělen do několika fází. Od 2.1.2018 do 8.1.2018 probíhala tvorba technické dokumentace testovaných funkcionalit. Vývoj a implementace proběhl v předpokládaném termínu od 9.1.2018 do 19.1.2018. Na základě studia dokumentace

probíhala test analýza. Samotné testování začalo podle plánu 5.2.2018 ihned po nasazení na testovací prostředí. Testování včetně reportování nalezených defektů probíhalo do 23.2.2018.

Všechny testovací scénáře byly otestovány a nalezené defekty zaznamenány. Výsledky exploratorních testů byly předány k další analýze. Na základě výsledků testů budou funkcionality přepracovány. Projekt se tedy vrátil do fáze návrhu technického řešení především kvůli implementaci nedostatečných bezpečnostních opatření zjištěných exploratorním testováním.

6 Závěr

V diplomové práci je řešen návrh plánu testování, ve kterém je definovaná strategie testovacího procesu reálné aplikace. Součástí praktické části je vytvoření specifikace k dostupným případům užití, podle kterých je následně proveden návrh testovacích scénářů. Na základě provedených testů jsou zaznamenány nalezené chyby a sepsána doporučení k jejich odstranění.

V první části práce byla shrnuta a popsána teoretická východiska, která tvořila předpoklady pro vytvoření praktické části práce. V úvodu byl popsán životní cyklus softwaru a shrnuty jednotlivé modely vývoje používané v praxi. Další část pak byla věnována metodikám návrhu vývoje softwaru, přičemž byl kladen důraz hlavně na proces testování, který je jejich součástí. Samotný proces testování, typy testů a techniky návrhu testovacích scénářů včetně zaznamenání chyb a zajištění jejich opravy byly popsány v dalších kapitolách teoretické části. Poslední kapitola teoretické části byla věnována plánování testování, poznatky zjištěné studiem této problematiky byly využity při plánování testování reálné aplikace v praktické čísti diplomové práce.

Vlastní práci tvořil na základě popisu testované aplikace a jejích funkcionalit návrh plánu testování. Dle specifikované strategie v plánu testování byly vytvořeny specifikace pro každý testovaný případ užití. Bylo využito dvou odlišných technik návrhu konkrétních testů. První technika se zabývala návrhem testů podle vytvořené specifikace, tyto testy se snažily pokrýt hlavní i alternativní scénáře testovaných funckionalit aplikace. Druhou technikou využitou při návrhu testů byla technika založená na zkušenostech. Pomocí této techniky byly identifikované rizikové oblasti dané aplikace, které byly podrobeny exploratornímu testování. Nalezené chyby byly v souladu s plánem testování zaznamenány v aplikaci Trello.

V závěru práce byly zhodnoceny dosažené výsledky jak v návrhu plánu testování, tak výsledky provedených testů. Chyby nalezené funkčním testováním byly popsány a přiřazeny vývojovému týmu pro zajištění opravy. Výsledky provedených exploratorních testů byly reportovány analytickému týmu. Součástí reportu byla doporučení možných opatření vedoucích k eliminaci nalezených nedostatků aplikace.

7 Seznam použitých zdrojů

1. What are the Software Development Life Cycle (SDLC) phases? ISTQB EXAM CERTIFICATION [online]. 2017 [cit. 2018-02-02]. Dostupné z: http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/
2. PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. Programování. ISBN 8072266365
3. SOMMERVILLE, Ian. *Softwarové inženýrství*. Brno: Computer Press, 2013. ISBN 9788025138267
4. BRIAN HAMBLING (EDITOR) a PETER MORGAN .. [ET AL.]. *Software testing: an ISTQB-ISEB foundation*. 2nd ed. London: British Computer Society, 2010. ISBN 9781906124762
5. ŠMÍD, Vladimír. *Životní cyklus informačního systému* [online]. [cit. 2018-02-02]. Dostupné z: https://www.fi.muni.cz/~smid/mis-zivcyk.htm
6. BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6
7. What is Incremental model- advantages, disadvantages and when to use it? ISTQB EXAM CERTIFICATION [online]. 2017 [cit. 2018-02-03]. Dostupné z: http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/
8. Role RAD platformy v podnikové architektuře. Business World [online]. 2018 [cit. 2018-02-07]. Dostupné z: http://businessworld.cz/aplikace/role-rad-platformy-v-podnikove-architekture-13917
9. What is Spiral model - advantages, disadvantages and when to use it? ISTQB EXAM CERTIFICATION [online]. 2017 [cit. 2018-02-07]. Dostupné z: http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/
10. Spirálový model. Testování Softwaru [online]. [cit. 2018-02-11]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/
11. BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7
12. ŠTĚDROŇ, Bohumír. Manažerské řízení a informační technologie. Praha: Grada, 2007. ISBN 978-80-247-2052-4
13. Rational Unified Process: Best Practices for Software Development Teams. IBM [online]. [cit. 2018-02-11]. Dostupné z: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
14. Principy stojící za Agilním Manifestem. Agile Manifesto [online]. 2001. [cit. 2018-02-16]. Dostupné z: http://agilemanifesto.org/iso/cs/principles.html
15. Extreme Programming: A gentle introduction [online]. 2013. [cit. 2018-02-16]. Dostupné z: http://www.extremeprogramming.org/

16. BECK, Kent. a Cynthia. ANDRES. Extreme programming explained: embrace change. 2nd ed. Boston, MA: Addison-Wesley, 2005. ISBN 978-0321278654
17. BREWER, John. Introduction to Extreme Programming [online]. [cit. 2018-02-16]. Dostupné z: http://www.jera.com/techinfo/xphandout.pdf
18. ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8
19. LUKEŠ, Pavel. Zátěžové testy podnikových portálů. Praha, 2015. Diplomová práce Bankovní institut vysoká škola.
20. BEIZER, Boris. Software testing techniques. 2nd ed. New Delhi: Dreamtech, 2003. ISBN 978-81-772-2260-9
21. Certifikovaný tester: Učební osnovy pro základní stupeň [online]. 2017 [cit. 2018-02-17]. Dostupné z: http://castb.org/cz/
22. BLACK, Rex. Managing the testing process: practical tools and techniques for managing software and hardware testing. 3rd ed. Indianapolis, MN: Wiley, 2009. ISBN 978-047040415
23. JORGENSEN, Paul. Software testing: a craftsman's approach. 3rd ed. Boca Raton: Auerbach Publications, 2008. ISBN 0-8493-7475-8
24. PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. Jak testuje software Microsoft. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5
25. HLAVA, Tomáš. Testování softwaru [online]. [cit. 2018-02-18]. Dostupné z: http://testovanisoftwaru.cz/
26. MICHÁLEK, Lukáš. Použití metod testování softwaru v praxi. Praha, 2006. Bakalářská práce VŠE Praha.
27. ČÁPKA, David. UML – Use Case Specifikace. ITnetwork [online]. [cit. 2018-02-18]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-specifikace-diagram
28. SPONSOR a SOFTWARE ENGINEERING TECHNICAL COMMITTEE OF THE IEEE COMPUTER SOCIETY. IEEE standard for software test documentation. Rev. ed. New York: Institute of Electrical and Electronic Engineers, 1998. ISBN 0738114448