Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Testování softwaru

(Automatizované testování softwaru)

Bakalářská práce

Autor: Tomáš, Burda
Studijní obor: AI3

Vedoucí práce: doc. RNDr. Petra Poulová, Ph.D.

Odborný konzultant: Ing. David Petrík
Systemart s. r. o.

Hradec Králové červen 2017

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 20.4.2017 Tomáš Burda

Poděkování:

Děkuji vedoucí bakalářské práce doc. RNDr. Petře Poulové, Ph.D. za metodické vedení práce a Ing. Davidu Petríkovi ze společnosti Systemart s. r. o. za odborný dohled a užitečné rady.

Anotace

Bakalářská práce si klade za cíl seznámit čtenáře srozumitelnou formou s problematikou testování softwaru. Práce je rozdělena na teoretickou a praktickou část. Teoretická část obsahuje základní informace o testování softwaru, včetně popisu způsobů testování. Tyto způsoby se rozlišují podle toho, jak přistupují k aplikaci nebo v jaké fázi životního cyklu softwaru se provádějí. Na závěr je významná část věnována automatizovanému testování, včetně popisu nejpoužívanějších nástrojů k automatizaci testování webových, mobilních a desktopových aplikací.

Praktická část práce se zaměřuje na proces testování softwaru ve společnosti Systemart s. r. o., která vyvíjí aplikaci Výkaz práce. Závěr je věnován analýze procesu testování, výběru vhodných nástrojů k automatizaci a samostatné implementaci automatizovaných testů pro webovou aplikaci.

Annotation

Title: Software testing

This bachelor thesis aims to introduce the reader an issue of software testing understandable way. The thesis is divided into theoretical and practical part. The theoretical part contains basic information about software testing, including a description of test methods. These methods are distinguished by how they access the application, or at what stage of the software life cycle are they carried out. Finally, a significant part deals with automated testing, including a description of the most commonly used tools for automated testing of web, mobile and desktop applications.

The practical part focuses on the process of software testing in the company Systemart Ltd., which develops the application Výkaz práce. The conclusion deals with analyzing testing process, selecting the appropriate tools for automation and separated implementation of automated testing for the web application.

Obsah

Seznam obrázků

Seznam grafů

1 Úvod

Ve své bakalářské práci jsem se na základě vlastních zkušeností rozhodl zabývat testováním softwaru. Zkušenosti s testováním jsem postupně získával ve společnosti Systemart s. r. o. Jelikož společnost klade velký důraz na kvalitu softwaru, začali jsme spolupracovat na zlepšení celého procesu testování. Za řízení tohoto procesu a jeho zlepšení ve firmě nesu veškerou zodpovědnost. Tato skutečnost podpořila mé rozhodnutí o výběru tématu a umožnila spojit bakalářskou práci přímo s praktickým využitím. Veškeré nabyté znalosti a zkušenosti budou tedy aplikovány na reálném případě.

Testování softwaru je nepostradatelnou součástí každého životní cyklu vývoje softwaru a umožňuje zlepšování kvality vyvíjených systémů. Z toho důvodu byly vytvořeny různé testovací postupy zabývající se dosažením požadované kvality produktu.

V současné době, kdy vznikají nové technologie, platformy a programovací jazyky, se testování softwaru stále rozvíjí a mění. Aktuální témata posledních let jsou zejména testování mobilních aplikací, testování využívající cloud, virtualizace nebo využití automatizačních nástrojů.

Prvním cílem této bakalářské práce je seznámit čtenáře s testováním softwaru, přičemž významná část je věnována automatizovanému testování. Druhým cílem je zlepšit proces testování nad konkrétním projektem.

Práce je rozdělena na teoretickou a praktickou část. Teoretická část se věnuje základním pojmům, které jsou nezbytné pro porozumění této oblasti. Ve velké míře se jedná o představení technik testování a významu přístupů vývoje softwaru pro proces testování. V dalších kapitolách je popsán význam automatizovaného testování, včetně nejpoužívanějších nástrojů k testování webových, mobilních a desktopových aplikací.

Praktická část seznámí čtenáře s problematikou testování softwaru aplikace Výkaz práce. Poslední kapitoly se proto věnují tomuto projektu, včetně analýzy životního cyklu vývoje softwaru. Závěr bakalářské práce je věnován výběru vhodných nástrojů k automatizaci a samostatné implementaci automatizovaných testů pro webovou aplikaci.

I. Teoretická část

2 Testování softwaru

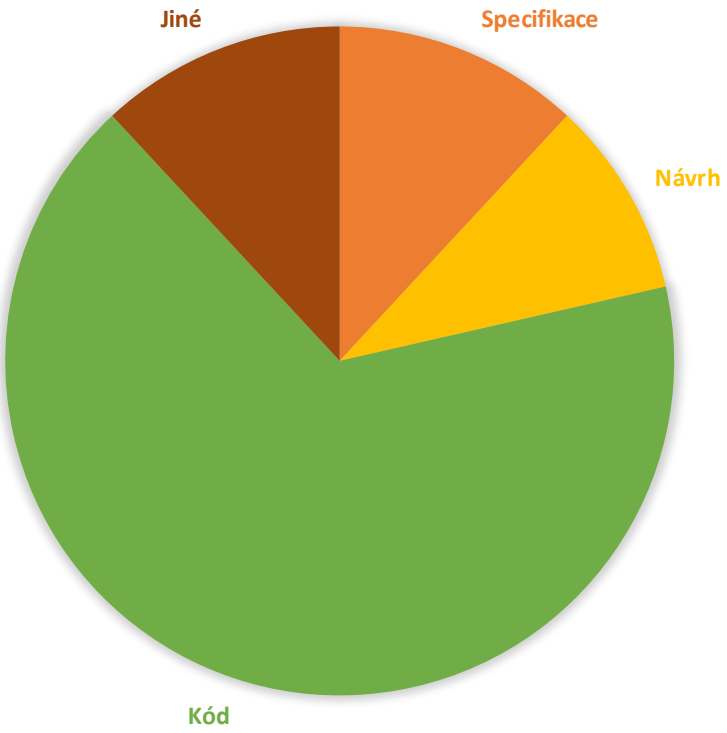
Testování softwaru je důležitá práce, která je v dnešní době vzhledem k rozsahu a složitosti softwaru naprosto nezbytnou součástí v každém životním cyklu vývoje softwaru. Rizika selhání softwaru jsou příliš vysoká a mohou mít až katastrofické následky. Pomocí dále popsaných způsobů testování lze rizika eliminovat na minimum, a tím zajistit požadovanou kvalitu softwaru.

Testování softwaru se provádí vůči specifikaci produktu, která je dohodou v rámci vývojového týmu. Specifikace podrobně popisuje produkt: jak bude vypadat, jak se bude chovat, co bude dělat nebo co dělat nebude. Je zapotřebí brát v úvahu, jaké požadavky na produkt budou mít koncoví uživatelé. Musí se proto zvážit, kdo a jak bude vyvíjený software používat, a netestovat ho pouze jako porovnání se specifikací. Testování slouží jako nástroj k získávání informací o kvalitě softwaru, přičemž nalezené softwarové chyby jsou vedlejším produktem této činnosti [2]. Softwarovou chybu definoval Ron Patton následovně [1]:

„O softwarové chybě hovoříme právě tehdy, pokud je splněna jedna nebo více z následujících podmínek:

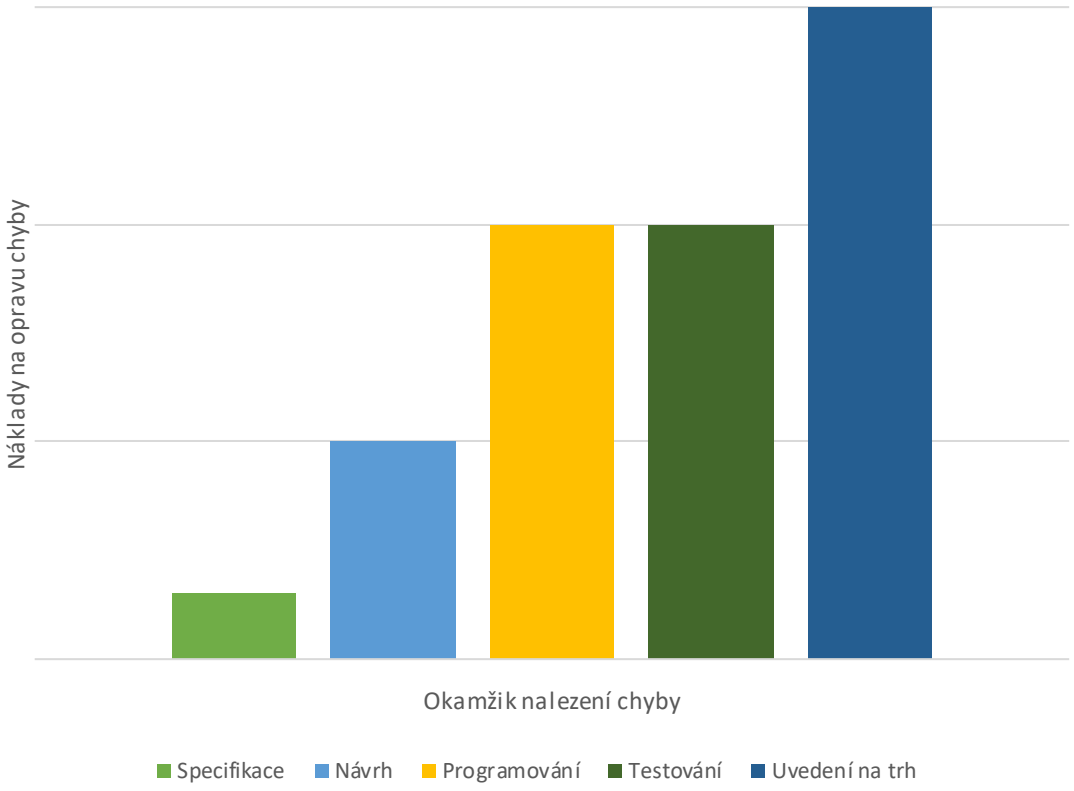
1. *Software nedělá něco, co by podle specifikace produktu dělat měl.*
2. *Software dělá něco, co by podle údajů specifikace produktu dělat neměl.*
3. *Software dělá něco, o čem se produktová specifikace nezmiňuje.*
4. *Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.*
5. *Software je obtížné srozumitelný, těžko se s ním pracuje, je pomalý, nebo – – podle názoru testera softwaru – jej koncový uživatel nebude považovat za správný."*

Graf 1: Rozložení příčin softwarových chyb uvádí rozložení příčin chyb, kde jednoznačně převládají chyby s původem ve zdrojovém kódu. Před několika lety byla situace úplně odlišná, hlavním příčinou byla specifikace. Tato změna je způsobena zejména dodržováním norem a využíváním stále lepších nástrojů během analýzy a návrhu systému. [2]



Graf 1: Rozložení příčin softwarových chyb
Zdroj: [2]

Vývoj softwaru je obvykle plánován podle některé z metodik plánování softwaru. Chyby může tester odhalit od zahájení vývoje přes plánování, programování, testování, až po okamžik, kdy je software zveřejněn. *Graf 2: Náklady na opravu chyb v průběhu času* ukazuje, jak je důležité začít s testováním co nejdříve, tedy už ve fázi specifikace. Platí zde, že čím dříve je chyba odhalena, tím jsou nižší náklady na její odstranění. [2]



Graf 2: Náklady na opravu chyb v průběhu času
Zdroj: [1]

Testováním softwaru nelze zajistit, aby výsledný software byl zcela bez chyb, a to ani u nejjednodušších programů. Počet chyb lze pouze eliminovat na minimum. Důvody jsou následující:

- Počet možných vstupů softwaru je příliš velký.
- Počet možných výstupů softwaru je příliš velký.
- Počet možných cest, které vedou skrze software, je příliš velký.
- Specifikace softwaru je subjektivní. Vždycky můžeme říct, že jediná chyba je pouze v očích pozorovatele.

Každý softwarový projekt si musí zvolit nějakou efektivní hladinu testování. Jde o to nalézt vhodný kompromis mezi množstvím provedených testů (nákladů na testování) a počtem odhalených chyb. [1]

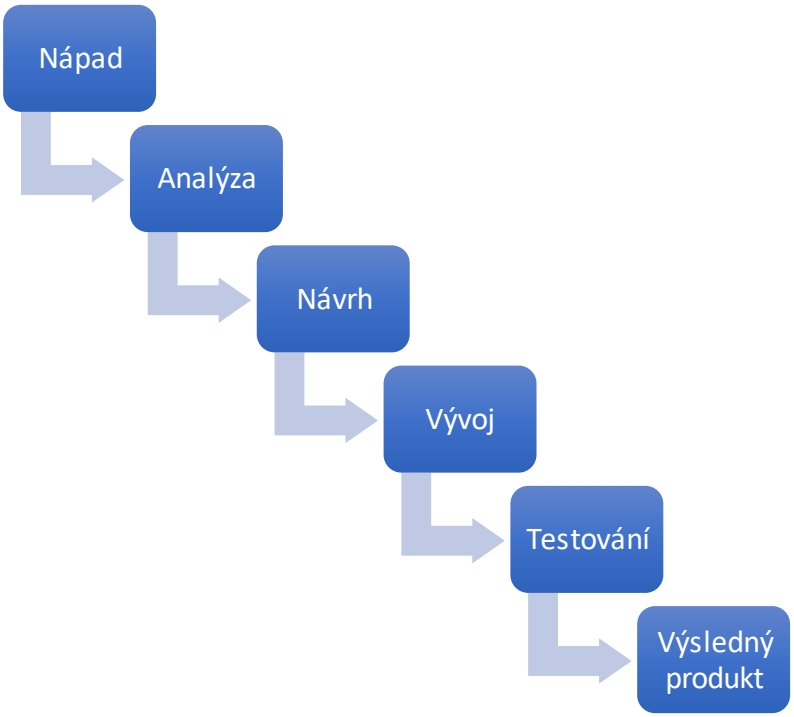
2.1 Modely životního cyklu softwaru

Tester softwaru musí alespoň zhruba porozumět celkovému procesu vývoje softwaru, aby mohl správně přizpůsobit různé postupy testování. Do vývoje nového softwaru mohou být zapojeny desítky až tisíce členů vývojového týmu, každý má jinou roli, ale všichni spolupracují podle stanoveného modelu.

Všechny modely mají své výhody i nevýhody. Každý z nich obsahuje vlastní metodiku, jak zajistit dostatečnou kvalitu softwaru. Výběr správného modelu životního cyklu je klíčový pro budoucí úspěch celého projektu. V současné době je možné použít celou řadu modelů životního cyklu softwaru, ale většina z nich vychází z definic vodopádového nebo spirálového modelu. [1][3]

2.1.1 Model vodopádu

Vodopádový model je jednoduchý a snadno pochopitelný, proto se využívá k výukovým účelům. Vychází ze sekvenčního přístupu k jednotlivým fázím a je charakteristický tím, že do další fáze lze vstoupit, pokud je předchozí fáze dokončena a uzavřena. Jednotlivé fáze tohoto modelu popisuje *Obrázek 1: Model vodopádu*.



Obrázek 1: Model vodopádu
Zdroj: Vlastní zpracování

Vodopádový model bývá využíván u menších projektů, kde se nepočítá s pozdějšími změnami funkcionality nebo vlastností celého výsledného softwaru. Cílem je ještě před zahájením programování rozpracovat všechny neznámé a popsat všechny detaily.

Hlavní nevýhodou tohoto modelu je fakt, že nelze dokončit jednu fázi a zahájit další bez toho, aniž by se k ní dalo v budoucnu vrátit. Z hlediska testování je nevýhodou, že testování probíhá až na konci vývojového cyklu. Toto může mít za následek, že chyba vzniklá na začátku cyklu je odhalena, až když je produkt připravený k umístění na trh.

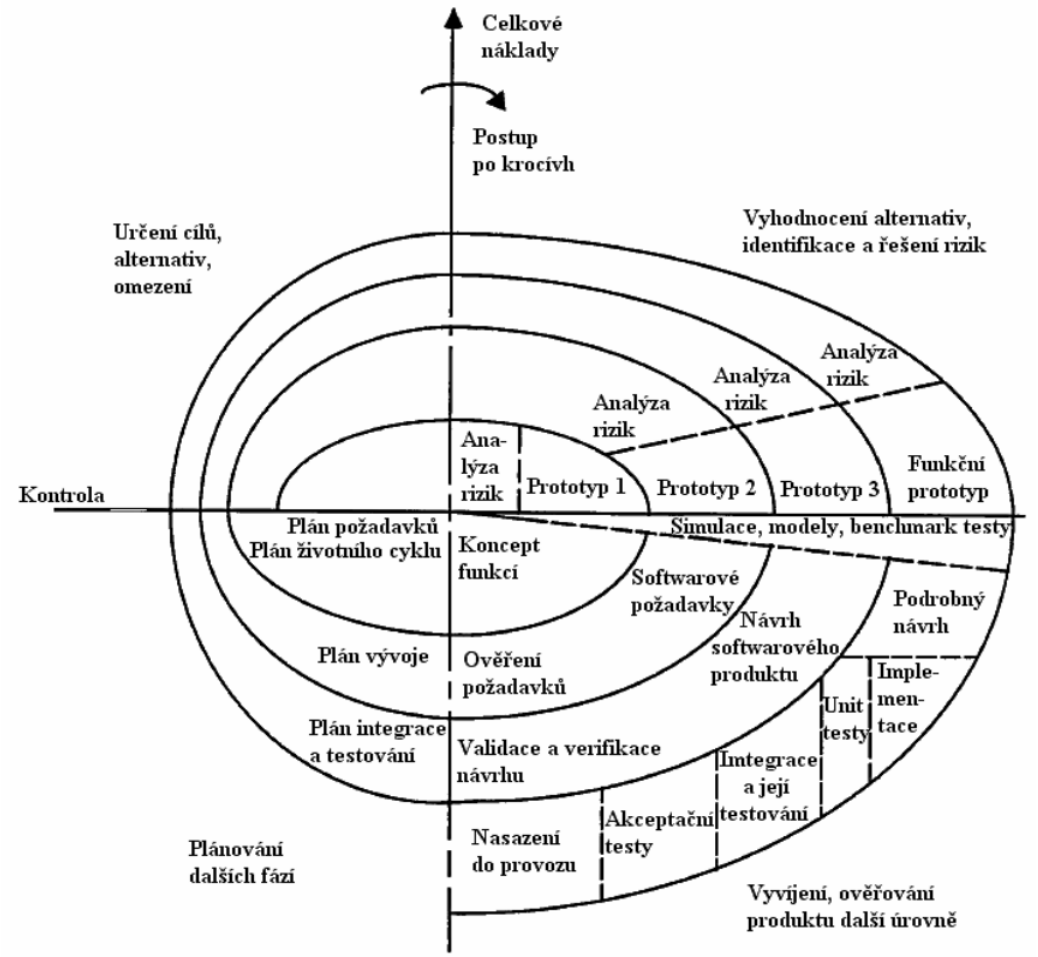
V dnešní době vzniklo z původního modelu několik různých variant, ve kterých se mohou jednotlivé fáze částečně překrývat a v případě potřeby je možné se vrátit o jeden krok zpět. [1][4]

2.1.2 Spirálový model

Spirálový model je pro vývoj softwaru velice efektivní a velmi dobře pokrývá největší nedostatky vodopádového modelu. Model probíhá v několika krocích, které se neustále opakují, dokud není výsledný software zcela hotov. Ze začátku se vývoj provádí na základě hrubé specifikace požadavků, které jsou v pozdějších fázích upřesňovány.

Při každém průchodu spirálou se provádí následující kroky:

1. Určení cílů, alternativ a omezení
2. Rozpoznání a řešení rizik
3. Vyhodnocení alternativ
4. Vývoj a testování aktuální úrovně
5. Plánování další úrovně
6. Rozhodnutí o postupu na další úroveň



Obrázek 2: Spirálový model
Zdroj: [5]

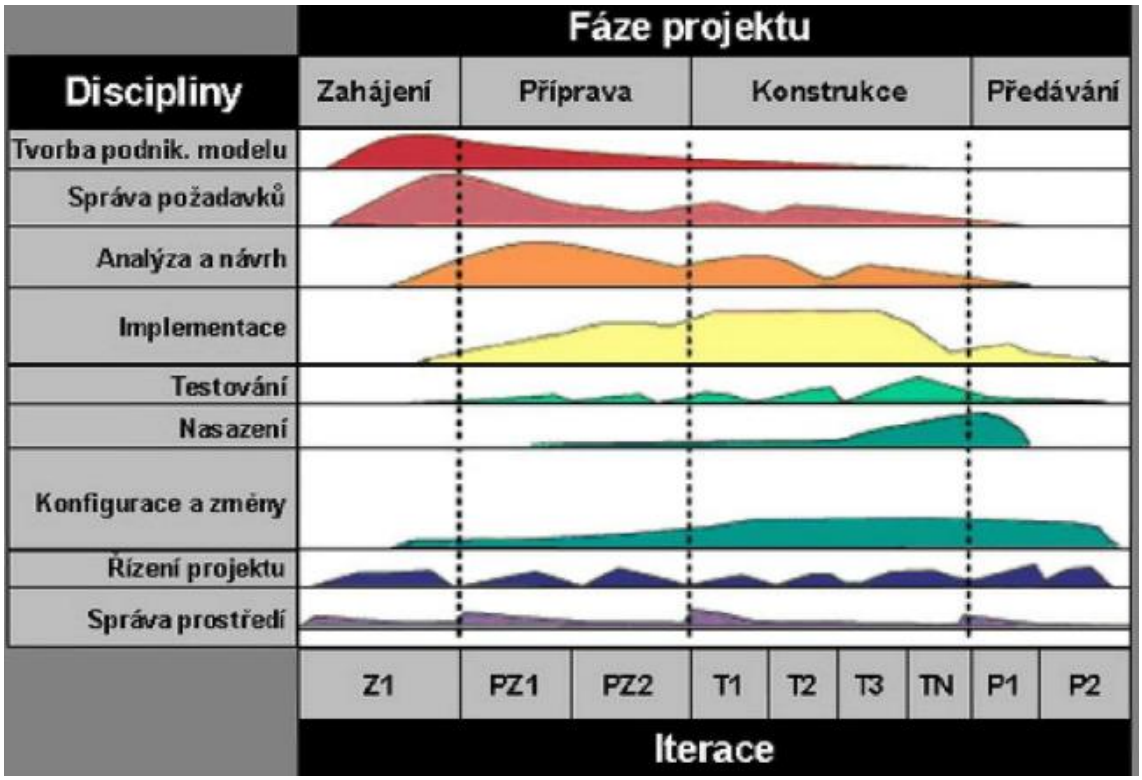
Velkou výhodou tohoto modelu je, že se testování provádí v každém průchodu spirálou. Software je tak testován pravidelně už od počátku vývoje, a tím dochází k včasnému odhalení chyb. Spirálový model je vhodný i pro nasazení automatizovaných testů, které je nutné pouze upravit dle aktuální verze. [1][3]

2.1.3 Rational Unified Process

Rational Unified Process (RUP) je objektově orientovaný iterativní přístup k životnímu cyklu softwaru, který náleží do skupiny přístupů řízených případy užití. Model obsahuje čtyři základní fáze, kdy každá z nich je rozdělena na několik dalších iterací. Před zahájením nové iterace musí být splněna kritéria iterace předchozí.

- **Fáze zahájení** – Definuje účel a rozsah projektu.
- **Fáze projektování** – Analyzuje požadavky zákazníka.
- **Fáze realizování** – Tvorba zdrojových kódů.
- **Fáze předání** – Projekt je předán zákazníkovi nebo do dalšího cyklu.

Obrázek 3: Rational unified process znázorňuje, jaký objem pracovních kapacit je rezervovaný v danou chvíli na daný proces v konkrétní fázi projektu.



Obrázek 3: Rational unified process
Zdroj: [5]

Testování probíhá během celého modelu, nejintenzivněji však těsně před předáním produktu zákazníkovi. Je zde prostor testy neustále rozvíjet a plánovat do dalších iterací. [6]

2.1.4 Agilní vývoj

Agilní přístup je orientován na jednotlivé členy týmu, jejich schopnosti a interakci. Umožňuje rychlý vývoj softwaru a je schopný rychle reagovat na změnu požadavků v průběhu cyklu. Naopak do pozadí ustupuje striktní dodržování procesů nebo používání daných nástrojů.

Agilní přístup využívá kratších iterací, kdy jednotlivé funkční celky jsou dodávány v časovém intervalu většinou 2 až 4 týdnů. Požadavky se implementují podle priority, kterou určuje zákazník.

Mezi agilních přístupy patří scrum, extrémní programování nebo vývoj řízený testy. [2][3]

2.1.4.1 Vývoj řízený testy

Vývoj řízený testy (z anglického „Test-Driven Development“ – zkratka TDD) znamená, že se nejdříve naprogramují jednotkové testy, a to ještě před dokončením návrhu a zahájením programování.

Testy se nejprve spustí tak, aby se ověřilo, že žádný z nich neprojde. Pokud by tomu tak nebylo a test skončil úspěšně, znamenalo by to, že je špatně napsaný, nebo končí úspěšně vždy. Programovaný kód je následně napsán tak, aby všechny testy prošly úspěšně.

Hlavní výhodou je možnost ihned testovat kód po jeho dopsání. Nevýhodou může být přehlednost a zvýšení objemu kódu, nebo skutečnost, že i testy mohou obsahovat chyby. [7]

2.2 Způsoby testování

2.2.1 Manuální a automatické testování

Testy lze rozlišit podle toho, zda jsou prováděny softwarem, nebo člověkem. Pokud test vyžaduje lidské ohodnocení a úsudek, je vhodnější využít manuální testování. Pro opakované spouštění velkého počtu testů s velkým množstvím generovaných dat je vhodnější použít automatické testování. Tomuto tématu se více věnuje kapitola 3. [1]

2.2.2 Statické a dynamické testování

Na základě toho, zda je k testování nutné spuštění aplikace, rozlišujeme statické a dynamické testy.

Statické testy nevyžadují běh aplikace. Využívají se v prvních fázích vývoje, kdy ještě není k dispozici funkční aplikace. Jsou vhodné na revizi programového kódu nebo pro kontrolu specifikace požadavků.

Dynamické testy vyžadují spuštění aplikace. Využívají se v pozdějších fázích vývoje, kdy už je k dispozici spustitelná aplikace. [1]

2.2.3 Testování černé a bílé skříňky

Testování černé skříňky představuje techniku, kdy tester nemá přístup k programovému kódu. Software si lze představit jako černou skříňku, jejíž obsah není zvenčí viditelný. Neznáme, jak přesně systém pracuje s daty, pouze sledujeme, jaký výsledek získáme po vložení vstupních dat. Software v tomto případě bývá testován pomocí testovacích scénářů, které definují, jak má tester postupovat při testování. Výhodami této techniky jsou rychlost a snadnost, neboť tester nemusí mít znalosti konkrétního programovacího jazyka. Nevýhodou pak je, že testovací scénář nemusí pokrýt potenciální chyby.

U testů bílé skříňky naopak známe vnitřní strukturu softwaru a máme k dispozici zdrojový kód. Můžeme lépe otestovat všechny průchody zdrojovým kódem, zadání neočekávaných vstupních hodnot nebo možnost odhalit nežádoucí části kódu. Tato technika bývá náročnější, neboť je důležité analyzovat zdrojový kód a porozumět mu, zároveň tím ale získáme vyšší kvalitu kódu výsledného produktu.

Můžeme se setkat i s kombinací obou kategorií, která bývá nazývaná jako testy šedé skříňky. Jedná se o situaci, kdy software testujeme přes jeho uživatelské rozhraní a výsledky operací ověřujeme pomocí dotazů do databáze. [1][2][3]

2.2.4 Konfirmační a regresní testování

Konfirmační testy se využívají pro kontrolu nových funkcí nebo vlastností softwaru a používají se ve všech etapách testování. Pro jejich správné provedení je nutná dokumentace, která popisuje nové funkce a vlastnosti softwaru.

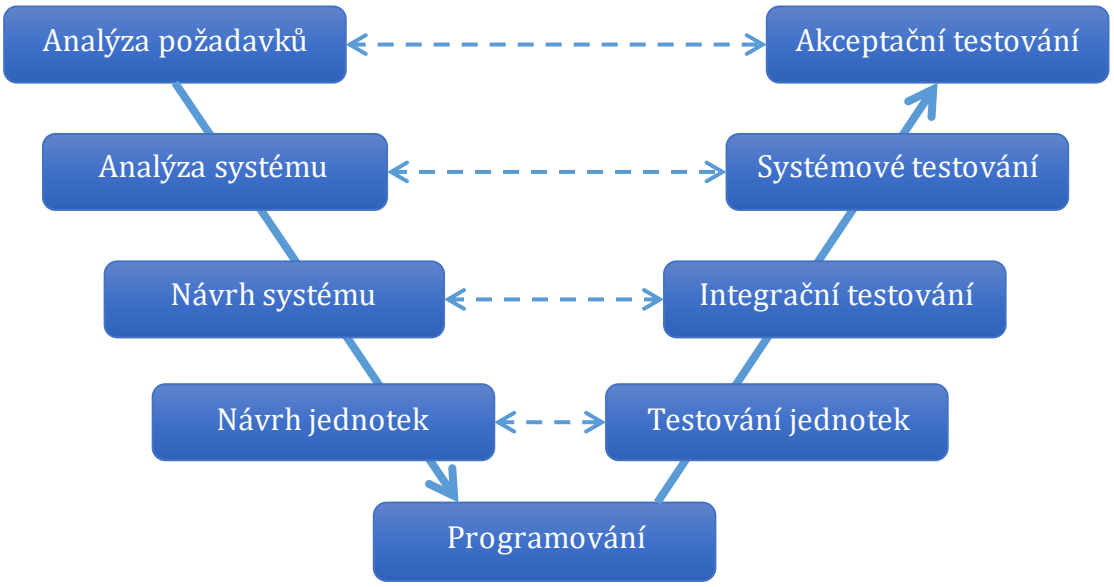
Regresní testy se využívají pro opakované testování funkcí a vlastností softwaru. Ideálně jsou prováděny vždy se změnou softwaru. Jejich účelem je ověřit, zda provedené změny nebo implementace nových vlastností softwaru nepřinesly žádné další chyby v dosud fungujících částech aplikace. Regresní testy jsou velmi rozšířeny a je vhodné je automatizovat. [2]

2.3 Úrovně testování

Během vývoje je testování prováděno na různých úrovních. Podle toho, v jaké úrovni a s jakým časovým odstupem od napsání zdrojového kódu se testování provádí, jej rozdělujeme do následujících úrovní:

1. Testování jednotek (Unit testing)
2. Integrační testování (Integration testing)
3. Systémové testování (System testing)
4. Akceptační testování (Acceptance testing)

Testování prováděné na jednotlivých úrovních je podle V-modelu, který vychází z modelu vodopádu. Významným rysem V-modelu je účast testování od počátku projektu, neboť výstup z každé fáze prochází ověřením. [1][2]



Obrázek 4: V-model
Zdroj: [4]

2.3.1 Testování jednotek

Testování jednotek lze chápat jako proces testování co možná nejmenší testovatelné součásti softwaru – tzv. jednotky. Nejčastěji testování provádí sám programátor, který si ověří správnost své implementace. Podle toho, zda se jedná o objektově orientovaný či procedurální programovací jazyk, mohou jednotky být funkce, procedury, metody nebo třídy. Cílem testování jednotek je otestovat každou jednotku nezavisle na ostatních a ověřit, zda její chování je správné.

Testování jednotek je výhodné pro automatizaci a nutí vývojáře se nad svým kódem více zamýšlet. Ovšem zachytí pouze chyby související s přímým chováním jednotek. [2]

2.3.2 Integrační testování

Cílem integračního testování je nacházet chyby vzniklé propojením a následnou interakcí jednotlivých jednotek (tzv. modulů). Integrace probíhá tak, že do systému jsou postupně přidávány další moduly, které musí být jednotkově otestovány, a zároveň nesmí narušit stabilitu dosud fungujícího celku.

Integrační testy lze provádět na různých úrovních. Obecně se rozlišuje intrasystémové a intersystémové integrační testování. U intrasystémového

integračního testování jde především o integraci modulů do funkčního systému, zatímco v druhém případě jde o integraci dvou a více systémů do jednoho celku. [2]

2.3.3 Systémové testování

Systémové testy jsou používány až v pozdějších úrovních vývoje. Hlavním úkolem je ověření, zda systém splňuje požadavky zákazníka. Jde o poslední možnost nalézt a opravit chyby dřív, než by je objevil uživatel. Systém bývá procházen ve více kolech podle připravených scénářů, které simulují různé kroky, které v praxi mohou nastat. Nalezené chyby jsou opraveny a v dalších kolech je jejich oprava otestována.

Tato fáze zahrnuje celou řadu specifických typů testů. Běžně se provádí následující testy:

- **Bezpečnostní testy** – Zabývají se ověřováním, zda jsou data správná, neporušená a chráněná proti neoprávněnému přístupu zvenčí i zevnitř.
- **Testy robustnosti** – Zjišťují, jak systém pracuje s neočekávánými situacemi, chybnými vstupy nebo změnami prostředí, ve kterých běží.
- **Testy použitelnosti** – Kontrola, zda je použití systému pro uživatele srozumitelné a komfortní. Testuje se rozložení ovládacích prvků nebo snadné provádění často používaných funkcí.
- **Testy interoperability** – Ověřují, zda systém nebo část systému správně komunikuje se systémy od jiných dodavatelů. Testy jsou pouze zaměřeny na ověření funkčnosti vzájemné komunikace.
- **Testy spolehlivosti** – Zjišťují, jak dlouho je systém schopný pracovat bez toho, aniž by došlo k selhání.
- **Výkonnostní testy** – Měří výkon systému sledováním vybraných ukazatelů během různých scénářů.
    - **Zátěžové testy** – Simulují dlouhodobou zátěž pro ověření, zda systém zůstává použitelný a stabilní.
    - **Testy hraniční zátěže** – Ověřují chování systému během extrémní zátěže s cílem zjistit, jaká jsou omezení systému, nebo jaké je jeho chování, překračuje-li zátěž tuto hranici.

    - **Testy škálovatelnosti** – Jsou prováděny především u systémů, kde se v budoucnu očekává nárůst zátěže.
- **Testy přístupnosti** – Ověřují, zda s daným systémem mohou pohodlně pracovat i hendikepovaní uživatelé. Testy se zaměřují na velikosti písma a prvků, nastavení barev nebo na zvuková upozornění [3].
- **Testování dokumentace a lokalizace** – Zjišťuje, zda dokumentace neobsahuje gramatické chyby nebo zda všechny popsané postupy skutečně odpovídají aplikaci. Pokud je vyvíjený software vydáván ve více jazykových verzích, je nutná kontrola textových řetězců v uživatelském rozhraní [1].

Je-li v této fázi testování nalezen pouze malý počet chyb, nemusí to nutně hovořit o kvalitním softwaru. Vždy je na místě uvažovat spíše o nesprávně zvoleném a provedeném testování. [2]

2.3.4 Akceptační testování

Akceptační testování je prováděno na straně zákazníka. Cílem akceptačních testů je zjistit, zda produkt splňuje všechna kritéria, která jsou předem dohodnutá mezi zákazníkem a dodavatelem. Testování je zaměřené na scénáře zahrnující běžné aktivity z reálného používání. Nesplnění těchto kritérií může být důvodem k odmítnutí produktu.

V této fázi testování se můžeme setkat s pojmy alfa a beta testování, jejichž význam se může lišit. Obecně se alfa testování používá v prostředí dodavatele, zatímco beta testování probíhá u zákazníka. Druhé označení se používá pro ranou (alfa) a pozdější, stabilnější (beta) verzi systému. [2]

3 Nástroje pro testování

Testování softwaru je důležitý a nesnadný firemní proces. Zejména manuální testování je stereotypní, časově náročné a vyžaduje velký počet pracovníků v oddělení. Nevýhodou také je, že pracovníci časem ztrácejí pozornost z opakující se činnosti, a tím se zvyšuje riziko neodhalení některé závažné chyby v softwaru. Navíc je velmi obtížné otestovat několik stovek testů dané verze před uvolněním softwaru do produkce a je pravděpodobné, že již nezbude dostatek času na jejich opakování.

Opakování testů je důležité, neboť ověřuje, jestli odhalené chyby v předcházejících testech byly opraveny a jestli do softwaru nebyly zavlečeny žádné nové chyby. Tento proces se označuje jako zpětné neboli regresní testování. Těchto případů, kdy je potřeba opakovat jednotlivé testy, může během testování nastat velmi mnoho, což způsobí zdržení vydání aplikace. Řešením tohoto problému je nalézt vhodné nástroje pro testování softwaru, které ulehčí manuální testování a zvýší jeho efektivitu.

Výhody testovacích nástrojů pro provádění automatických testů jsou:

- **Rychlost** – Manuální testování člověkem je mnohonásobně pomalejší než v případě provedení automatických testů. Automatické nástroje jsou schopny provádět testy 10krát, 100krát, nebo i 1000krát rychleji.
- **Efektivita** – Během průběhu automatických testů se lze věnovat správě testování, plánování a psaní nových testovacích případů a provádění manuálních testů, které nelze automatizovat.
- **Správnost a přesnost** – Automatické testovací nástroje pracují vždy naprosto spolehlivě i po několika opakováních, a tudíž u nich nedochází k přehlédnutí chyby z nepozornosti. Výsledky testů jsou tedy vždy přesné.
- **Neúnavnost** – Na rozdíl od člověka se automatické testovací nástroje nikdy neunaví a prováděné testy nad softwarem mohou běžet opakovaně a nepřetržitě.

Všechny výše uvedené vlastnosti automatických nástrojů mohou vzbudit dojem, že stačí pouze pořízení takových nástrojů, které by zvládly veškerou práci za nás. Pouze bychom testy spustili a počkali si na výsledky. Tak jednoduché to ale bohužel není. Tyto nástroje se nehodí úplně vždy a u konkrétních případů se bez manuálního testování zcela neobejdeme. „*Automatické nástroje pro testování nemohou v žádném případě testery softwaru nahradit – pouze jim pomáhají odvádět jejich práci snáze a lépe.*“ [1] Předtím, než se rozhodneme pro automatické nástroje, je zapotřebí mít na mysli několik varování a omezení:

- Software se neustále vyvíjí, a to zejména tím, že jsou do něj přidávány nové funkce. Tyto změny mají velký vliv na již napsané testy, které by neproběhly správně. Automatizované testy se tedy musí napsat takovým způsobem, aby byla jejich úprava snadná a rychlá.
- Lidské oko a intuice se nedají ničím nahradit. Automatizované testy se řídí pouze podle naprogramovaného scénáře a nikdy nebudou testovat více, než co je od nich vyžadováno.
- Neměli bychom se spoléhat pouze na automatizaci. I přesto, že všechny automatizované testy proběhly úspěšně, neznamená to, že software je úplně bez chyb. Vždy se najdou nějaké chyby, které nejsou na první pohled patrné.
- Čas, který věnujeme práci s nástroji a automatizovanými testy, nesmí omezit důkladné manuální testování softwaru, jinak se může stát, že bychom žádné chyby nenašli.
- Některé nástroje mohou obsahovat chybu uvnitř své implementace. Chybu, nalezenou pomocí testovacího nástroje, je potřeba navodit znovu, ale tentokrát manuálně, tedy bez použití nástroje. Jedině tak můžeme prokázat, jestli se jedná o chybu v našem softwaru, která se opakuje, nebo zda je chyba přítomna v testovacím nástroji.

V následujících kapitolách jsou popsány nejrozšířenější nástroje, které jsou určeny k automatizovanému testování s podporou programovacího jazyka C#. [1][2][3]

3.1 Frameworky pro jednotkové testování

Jednotkové testování (z anglického „unit testing“) se zaměřuje, jak již bylo řečeno, na ověření správné funkcionality softwaru. Zahrnuje nástroje, činnosti a metodiky, které mají za cíl kontrolu funkčnosti malé části (tzv. jednotky) vyvíjeného softwaru. Jednotkou je myšlena část aplikace, nebo v případě objektově orientovaného programování třída či konkrétní metoda. K automatizaci testování je nutné psát testovací skripty zvané jednotkové testy. Jakmile jsou jednotkové testy k dispozici, je snadné je zcela automatizovat.

Výběr frameworků pro testování jednotek je závislý na platformě, na které je produkt vyvíjen. V dnešní době existuje na trhu celá řada frameworků pro různé platformy. Příklady těchto frameworků jsou NUnit (.NET), JUnit (Java), CppUnit (C++) nebo CUnit (C). Výhodou je integrace do vývojových prostředí (IDE), jako jsou Eclipse, NetBeans nebo Visual Studio, které usnadňují práci programátorům při vývoji softwaru. Vývojové prostředí většinou obsahuje editor zdrojového kódu, kompilátor, debugger atd. Také frameworky pro jednotkové testování umožňují v rámci jednoho vývojového prostředí vývojářům snazší psaní, spouštění, filtrování a prohlížení výsledků proběhlých testů. Frameworky umožňují využívat několik metod z třídy Assert, které ověřují správnost zdrojového kódu. Mimo spouštění testů ve vývojovém prostředí je lze spouštět také přímo z konzole, nebo obsahují vlastní uživatelské rozhraní, ve kterém jsou tyto testy přehledně zobrazeny. [8][9]

3.1.1 NUnit

NUnit je open source software, který je určen pro tvorbu jednotkových testů pro .NET framework a lze ho bez omezení využívat v bezplatných i komerčních aplikacích nebo knihovnách. Na jeho vývoji se podílí Charlie Poole, Rob Prouse, Simone Busoli, Neil Colvin a mnohočlenná komunita uživatelů. Z počátku vývoje vycházel z JUnit frameworku, který je určen pro programovací jazyk Java [10]. V následujících verzích byl NUnit kompletně přepsán s mnoha novými funkcemi, aby podporoval širokou škálu .NET platforem a aktuálně je k dispozici ve verzi 3.0.

Nejdůležitější vlastnosti frameworku:

- Testy lze spustit z konzole nebo ve Visual Studiu přes Test Adapter.
- Testy mohou běžet paralelně.
- Existuje silná podpora řízení testů.
- Podpora více platforem (např. .NET, Xamarin Mobile nebo Silverlight).
- Každý testovací případ může být přidán do jedné nebo více kategorií, což umožnuje selektivní spouštění.

Testovací metody a třídy jsou pro účely testování aplikace označeny pomocí speciálních atributů. Mezi používané atributy patří:

- **Test** – Je jedním ze základních způsobů označení metody jako test uvnitř třídy TestFixture.
- **TestFixture** – Označuje třídu, která obsahuje jeden nebo více testů a volitelné metody SetUp nebo TearDown.
- **SetUp** – Identifikuje metodu uvnitř TestFixture, která se automaticky spustí před každým jednotkovým testem. Využívá se k přípravě prostředí pro testy. Pokud SetUp metoda skončí chybou, následující test se nespustí a selže.
- **TearDown** – Označuje metodu uvnitř TestFixture, která je automaticky provedena po každém spuštěném testu. Pokud SetUp metoda proběhne bez chyb, je zaručeno spuštění TearDown metody.

Dalším důležitým prvkem, který hraje významnou roli v jednotkovém testování, je třída Assert. Třída Assert zahrnuje obsáhlou sadu statických metod, které ověřují pravdivost tvrzení. Tato tvrzení jsou předávána metodám pomocí parametrů. Pokud tvrzení selže, je hlášena chyba.

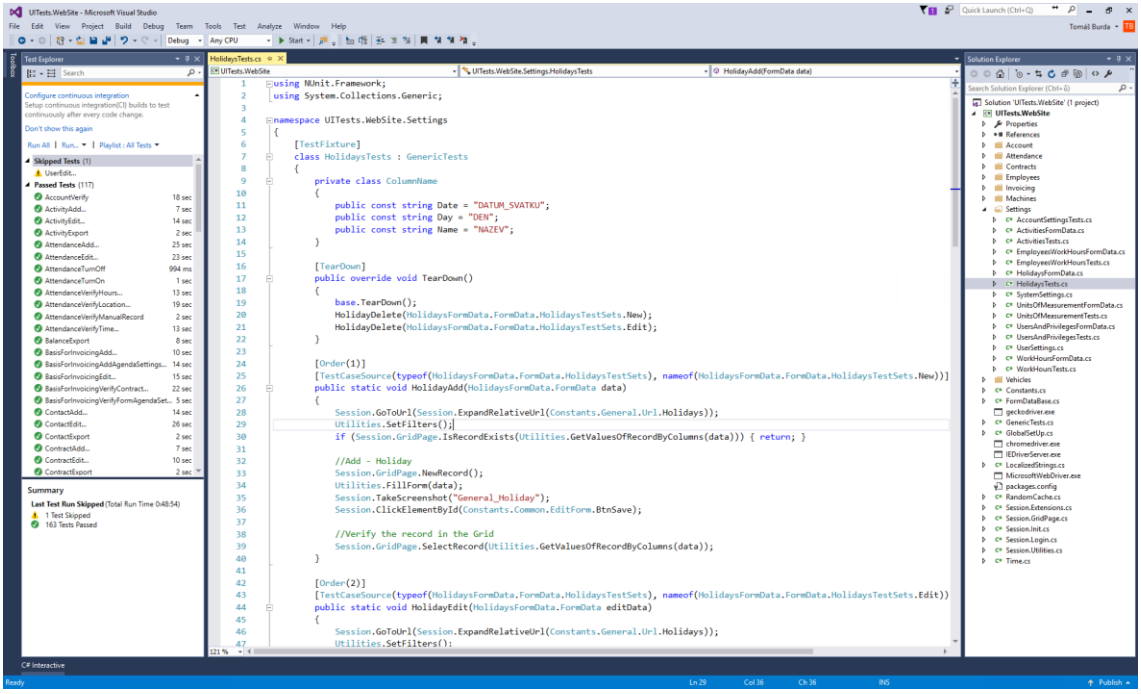
```
using NUnit.Framework;

namespace UITests.WebSite
{
        [TestFixture]
        public class FirstTest
        {
                [Test]
                public void MyFirstTest()
                {
                        Assert.True(IsOdd(3));
                }

                [TestCase(7)]
                [TestCase(3)]
                public void MyFirstTestCase(int number)
                {
                        Assert.True(IsOdd(number));
                }

                bool IsOdd(int number)
                {
                        return number % 2 == 1;
                }
        }
}
```

Obrázek 5: Ukázka jednotkového testu v NUnit
Zdroj: Vlastní zpracování

Výsledné jednotkové testy lze spustit v konzoli nebo v aplikaci s grafickým rozhraním.

- **Console Runner** – Program Nunit3-console.exe zobrazuje textový výpis a pro běh našich testů využívá příkazový řádek. Tento způsob je výhodný pro integraci do jiných systémů. Výsledky jsou uloženy ve formátu XML, který umožňuje vytvářet reporty a dále výsledky zpracovávat.

- **Visual Studio Test Adapter** – Test Adapter umožňuje spouštět testy uvnitř Visual Studia. Visual Studio s podporou Test Adapteru zobrazuje seznam jednotkových testů v Test Exploreru a nabízí uživatelům celou řadu funkcí pro správu testů, jako je spouštění, filtrování, vyhledávání, vyhodnocování nebo vytváření playlistů.



Obrázek 6: Visual Studio s podporou Test Adapteru
Zdroj: Vlastní zpracování

Tento framework lze rozhodně doporučit vývojářům bez větších zkušeností s jednotkovým testováním. NUnit 3.0 obsahuje kompletně aktualizovanou uživatelskou dokumentaci na svých webových stránkách a disponuje širokou komunitou uživatelů, kteří na internetu sdílí různé příklady a postupy. [11] [12]

3.1.2 XUnit.Net

XUnit.net je nejmodernější open source nástroj zaměřený na jednotkové testování v C#, F#, VB.NET a dalších jazycích .NET. Na jeho vývoji se podílejí vývojáři, kteří pracovali na NUnit 2.0. Zatímco většina testovacích frameworků si je velmi podobná, xUnit.Net přichází s minimalistickým, moderním a flexibilním přístupem. Mění se pojmenování atributů a metod. Již není potřeba definovat třídu, která

obsahuje testy. Testy mohou být v jakékoliv veřejné třídě. Dále chybí atributy SetUp a TearDown, které identifikují metody prováděné před testem a po každém testu. Tým xUnit.Net se domnívá, že tyto metody ztěžují přehlednost, ladění kódu a vedou ke zbytečnému spouštění kódu. Autoři nabízejí alternativu v podobě implementace konstruktoru bez parametrů a použití rozhraní IDisposable. XUnit.Net podporuje dva atributy, kterými lze identifikovat testovací metody:

- **Fact** – Atribut identifikuje metodu bez parametrů jako test.
- **Threory** – Atribut označuje parametrické testovací metody. Spolu s tímto atributem se definují sady dat, které jsou metodě předány. K tomu slouží atributy InlineData a MemberData.

```
using Xunit;

namespace Project
{
        public class FirstTest
        {
                [Fact]
                public void MyFirstFact()
                {
                Assert.True(IsOdd(3));
                }

                [Theory]
                [InlineData(7)]
                [InlineData(9)]
                public void MyFirstTheory(int number)
                {
                        Assert.True(IsOdd(number));
                 }

                  bool IsOdd(int number)
                {
                        return number % 2 == 1;
                 }
        }
}
```

Obrázek 7: Ukázka jednotkového testu v XUnit.Net
Zdroj: Vlastní zpracování

Výhoda frameworku je v paralelním provádění testů. Zatímco v NUnit není dosud paralelní provádění testů v rámci třídy implementováno, XUnit.net nemá v paralelismu žádné omezení. Tato vlastnost může ušetřit spoustu času, pokud máme mnoho testovacích případů.

Výsledné jednotkové testy lze spouštět přes konzoli, nebo v grafických aplikacích. Abychom mohli testy spouštět ve Visual Studiu přes Test Explorer, je potřeba stáhnout balíček xunit.runner.visualstudio. XUnit.Net vyvíjel vlastní grafický nástroj pro spouštění testů, který ale ve verzi 2.0 přestal podporovat.

Testovací framework díky své odlišnosti zaujal spíše zkušené vývojáře, a také ti mají problém si na tento odlišný přístup zvyknout. Framework může být složitý pro vývojáře bez větších zkušeností, nepřispívá tomu ani dokumentace, která není propracovaná, a z velké části srovnává své odlišnosti s NUnit. Vývojář je odkázán na komunitu uživatelů, kteří sdílí na internetu své příklady. [13]

3.1.3 Microsoft's Unit Testing Framework

Microsoft's Unit Testing Framework označovaný také jako MSTest, je dalším frameworkem pro tvorbu jednotkových testů pro jazyky .NET. Na jeho vývoji se podílí Microsoft a je přímo integrován do vyšších verzí Visual Studia, ve kterém lze napsané jednotkové testy psát, spouštět a ladit. Spouštění testů probíhá přes konzoli nebo Test Explorer. Testovací framework lze snadno používat, obsahuje základní funkce, které jsou velice podobné NUnitu. Pro označení testovacích tříd a metod se využívají atributy TestClass a TestMethod.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace SampleNetCoreUnitTests
{
	[TestClass]
	public class TestClass
	{
		[TestMethod]
		public void TestMethodPassing()
		{
			Assert.IsTrue(true);
		}

		[TestMethod]
		public void TestMethodFailing()
		{
			Assert.IsTrue(false);
		}
	}
}
```

Obrázek 8: Ukázka jednotkového testu v MSTest
Zdroj: [14]

Hlavní výhodou tohoto frameworku je jednoznačně integrace ve Visual Studiu a podpora ze strany Microsoftu. Bohužel framework obsahuje i několik nevýhod: oproti ostatním je pomalejší a těžko rozšiřitelný. Špatná rozšiřitelnost se projeví tehdy, pokud si vývojář chce přidat vlastní atributy. [14]

3.2 Nástroje pro automatizaci testování UI

Testování uživatelského rozhraní aplikace zajišťuje, aby aplikace odpovídala funkčním požadavkům a dosahovala vysoké kvality. Vysoká kvalita přináší vyšší pravděpodobnost, že bude aplikace úspěšně přijata a používána uživateli. Pro dosažení kvality se využívají nástroje, které simulují aktivitu skutečného uživatele a provádějí nad uživatelským rozhraním různé operace:

- Kliknutí na různé elementy
- Vložení nebo mazání hodnot do vstupních polí
- Získávání informací o elementech
- Ověření elementů

Nástroje pro automatizované testování uživatelského rozhraní lze rozdělit do několika skupin podle toho, o jaký typ aplikace se jedná:

- **Webové aplikace** – Aplikace poskytované uživatelům z webového serveru přes počítačovou síť. Uživatelé přistupují k aplikacím přes webový prohlížeč.
- **Mobilní aplikace** – Aplikace speciálně vytvořené pro chytré telefony, tablety a další mobilní zařízení.
- **Desktopové aplikace** – Aplikace speciálně vytvořené pro stolní počítače nebo notebooky.

Další dělení frameworků může dále být na základě ceny, skriptovacího jazyka, podpory více druhů prohlížečů, nebo podle toho, zda obsahuje vlastní aplikaci pro nahrání testů. [15]

3.2.1 Nástroje pro testování webových aplikací

3.2.1.1 Selenium

Selenium je open-source framework pro automatizované testování webových aplikací. Umožňuje psát testy v celé řadě programovacích jazyků, včetně C#, Java, Ruby, Python nebo Javascript. Výsledné testy lze spustit na většině moderních webových prohlížečů na platformách Windows, Linux a OS X.

Selenium v dnešní době obsahuje řadu nástrojů, díky kterým se stal asi nejrozšířenějším testovacím nástrojem webových rozhraní.

3.2.1.1.1 Selenium IDE

Selenium IDE je integrované vývojové prostředí pro Selenium skripty. Je integrované jako doplněk do webového prohlížeče Mozilla Firefox a umožňuje nahrávat, přehrávat, upravovat a ladit testy.

Hlavní výhodou Selenium IDE je jednoduché uživatelské rozhraní, které umožňuje vytvářet automatizované testy i uživatelům, kteří nemají žádné zkušenosti s programováním. Selenium zaznamenává jednotlivé akce uživatele, jako jsou vyplňování vstupních polí, kliknutí na ovládací prvky, otevření webové stránky na konkrétní URL adrese apod. Nahraný test lze ručně doplnit o další příkazy, které ověřují správnost hodnot. Výsledný test je uložený v HTML tabulce, ale pro další účely jej lze exportovat do libovolného programovacího jazyka, který je podporován.

Nevýhodou toho nástroje je, že Selenium IDE je dostupné pouze pro Mozilla Firefox, a tím je nemožné spouštět testy i v jiných prohlížečích.

3.2.1.1.2 Selenium WebDriver

Selenium WebDriver je nástroj pro psaní automatizovaných testů k testování webových aplikací. Poskytuje jednotné rozhraní, které pracuje s velkým počtem prohlížečů a umožnuje psát testy v téměř každém jazyce. Vzhledem k tomu lze jednou napsané testy spouštět na různých prohlížečích na různých platformách.

Cílem je poskytnout dobře navržené objektově orientované rozhraní, které lze snadno pochopit, a tím i výsledné testy byly snadno udržitelné. Selenium

WebDriver se stal nástupcem za Selenium RC a řeší jeho největší nedostatky. Největší výhody nástroje jsou:

- Není třeba spouštět server před spuštěním testů.
- Umožňuje použití cyklů, podmíněných operací a dalších programovacích technik.
- Podpora testování na zařízeních iPhone a Android.
- Umožnuje simulovat pohyb kurzoru nebo stisknutí kláves.
- Podpora funkcí jakou jsou navigace stránky nebo drag-and-drop.
- Přehledná dokumentace a velká komunita uživatelů.

```
//Najdi element podle ID
IWebElement element = driver.FindElement(By.Id("Button"));

//Klikni na element
element.Click();
```

Obrázek 9: Ukázka zdrojového kódu v Selenium
Zdroj: Vlastní zpracování

3.2.1.1.3 Selenium Grid

Selenium Grid je nástroj, který se specializuje na paralelní spouštění testů v různých webových prohlížečích, které běží na různých operačních systémech. Hlavní výhodou je zredukování doby trvání provádění sady testů a správa více prostředí z centrálního bodu. [16]

3.2.1.2 WatiN

WatiN je open-source nástroj pro testování webových aplikací, který je inspirovaný nástrojem Watir. K psaní testovacích skriptů se využívá jazyk C#, v kterém je i tento framework napsán. Jelikož tento nástroj přestal být udržován, nabízí v dnešní době pouze omezené funkce. Podporuje pouze Internet Explorer a Firefox, nelze spouštět testy ve více prohlížečích zároveň, nepodporuje vyhledávání elementů přes xpath. [17]

```
[Test]
public void SearchForWatiNOnGoogle()
{
        using (var browser = new IE("http://www.google.com"))
        {
                browser.TextField(Find.ByName("q")).TypeText("WatiN");
                browser.Button(Find.ByName("btnG")).Click();

                Assert.IsTrue(browser.ContainsText("WatiN"));
        }
}
```

Obrázek 10: Ukázka zdrojového kódu ve WatiN
Zdroj: [17]

3.2.2 Nástroje pro testování mobilních aplikací

3.2.2.1 Appium

Appium je open-source nástroj pro automatizované testování mobilních aplikací (nativních, hybridních a webových). Automatizované testy umožnuje spouštět na skutečných zařízeních, emulátorech nebo simulátorech a bez žádných zásahů do zdrojového kódu aplikace. V současné době, kdy každý vyvíjí mobilní aplikace nejméně pro dvě platformy, je další výhodou testování cross-platform, což znamená, že stejný test bude fungovat na různých platformách.

Appium je server, který je napsán v Node.js a implementuje Selenium WebDriver. Z toho důvodu je syntaxe testů totožná s testy pro webovou aplikaci a vývojáři se nemusí učit novou technologii. V současné době Appium podporuje platformy Android, iOS, Windows a Firefox OS. [18]

3.2.2.2 Calabash

Calabash je open-source nástroj vyvíjený a udržovaný společností Microsoft. Umožňuje psát a spouštět automatizované akceptační testy mobilních aplikací na platformách Android i iOS. Calabash se skládá z knihoven, které umožnují interakci testovacích kódů s nativními a hybridními aplikacemi. Interakce se skládá z řady uživatelských akcí, které mohou být typu gest, tvrzení nebo zachycení screenshotu.

Výhodou toho frameworku je zejména to, že je udržovaný Microsoftem, který poskytuje celou řadu dalších komerčních výrobků a služeb. Jednou z nich je Xamarin Test Cloud, který umožňuje spouštět testy na více než tisících reálných zařízení v cloudu.

Nevýhody mohou být omezení pouze na jeden programovací jazyk Ruby, nebo příprava speciálního targetu do buildu iOS aplikace. [19]

3.2.3 Nástroje pro testování desktopových aplikací

3.2.3.1 TestComplete

TestComplete je komerční nástroj vyvíjený společností SmartBear Software. Nabízí celou řadu nástrojů, které zjednodušují a zrychlují vytváření automatizovaných testů pro různé platformy. Umožnuje testovat webové, mobilní a desktopové aplikace a nabízí přístup k více než pěti stům reálných zařízení v cloudu. Pro psaní testů podporuje následující skriptovací jazyky: JavaScript, Python, VBScript, JScript, DelphiScript, C++Script a C#Script. Začínající uživatelé ocení jednoduché grafické rozhraní, které obsahuje funkce pro nahrávání a přehrávání testů bez znalostí skriptovacích jazyků.

Mezi další funkce, které TestComplete nabízí, patří výkonnostní testování, managment a monitoring testů nebo code review. Obdobné komerční řešení nabízí Telerik Test Studio nebo Ranorex. [20]

II. Praktická část

4 Aplikace Výkaz práce

Aplikace Výkaz práce je nástroj pro jednotlivce i velké firmy. Slouží k jednoduché evidenci docházky a práce na pracovišti, na cestách nebo z domova. Aplikace je postavena na platformách ASP .NET, Xamarin a Microsoft SQL server. Vyvíjena je společností Systemart, s. r. o., která byla založena v roce 2003, se sídlem společnosti v Hradci Králové a s dalšími pobočkami v Brně a Praze.

4.1 Popis aplikace

Celá aplikace v sobě zahrnuje několik agend, které slouží k evidenci a vyhodnocení zadaných údajů a parametrů. Uživatelé mají možnost vlastního nastavení systému podle svých potřeb. Funkcionalitu systému pokrývají následující agendy:

- **Docházka** – Docházkový systém umožňuje pracovníkům snadno zaznamenat příchody, odchody, absence i přestávky v práci. Díky tomu lze získat přehled o jejich přítomnosti na pracovišti a věrohodné podklady pro mzdy.
- **Rozpis práce** – V rozpisu práce mohou pracovníci rozepisovat čas strávený v práci na jednotlivé zakázky a jejich části. Spolu se zakázkami mohou zaznamenávat i další požadované údaje, které chceme sledovat. Díky možnosti nastavení systému získáme přehled o vykonávaných aktivitách a širokou škálu statistik.
- **Kniha jízd** – Pomocí knihy jízd je možné evidovat spolu s jízdními výkony vozidla také jízdy na jednotlivých zakázkách a pomocí nastavených sazeb za vozidlo sledovat i příslušné náklady.
- **Výkaz strojů** – Agenda umožňuje vykazovat práci strojů a nabízí základní funkce jako kniha jízd.
- **Zakázky** – Veškerou vykázanou práci zaměstnanců, vozidel i strojů lze evidovat na jednotlivé zakázky, popř. jejich členění. K těmto

výkonům lze na zakázky přidávat i ostatní náklady a výnosy a získat tak ucelený přehled o jejich ziskovosti.

- **Fakturace** – Agenda je určena především pro snadné a rychlé vytváření podkladů pro fakturaci z vykázané práce.

V současné době je Výkaz práce vyvíjen jako webová aplikace, ke které uživatelé přistupují prostřednictvím webového prohlížeče, přičemž jsou aktuálně podporovány Internet Explorer 11, Chrome, Firefox a Safari. Současně s webovou aplikací probíhá vývoj aplikace pro mobilní telefony s operačním systém Android a iOS a desktopové aplikace pro osobní počítače s operačním systémem Windows.

Celý systém si lze představit jako architekturu klient-server. Webová aplikace zde představuje server, který řídí připojené klienty. Mobilní zařízení, osobní počítače a pevný terminál jsou klienti, kteří se starají o sběr dat od pracovníků. Zařízení se neustále synchronizují, takže pracovníci i jejich vedoucí mají k dispozici vždy aktuální záznamy.

- **Webová aplikace** – Webová část aplikace je základním kamenem celého systému, bez kterého nelze Výkaz práce efektivně používat. Je zde hlavní uložiště dat, nabízí přehledné výstupy i možnost exportu, kromě zadávání jednotlivých záznamů umožňuje realizovat i veškerá nastavení dle potřeb uživatele.
- **Mobilní zařízení** – Jedná se o velice využívanou část celého systému. Aplikace vyniká svojí jednoduchostí. Spolu se zadáním docházky je zaznamenávána i poloha pracovníka, díky které lze mít přehled o pohybu pracovníka i o čase stráveném na jednotlivých lokalitách.
- **Osobní počítače** – Aplikace určená pro osobní počítače je snadno ovladatelná a rychlá, umožňuje kromě jiného automaticky generovat knihu jízd z navštívených lokalit. Navíc tato část funguje i v případě výpadku internetu. Po jeho připojení se data opět synchronizují s ostatními zařízeními.

- **Pevný terminál** – Je vhodný pro trvalá pracoviště. Jednotliví pracovníci se identifikují pomocí přidělených čipů, nebo pomocí otisků prstů. [22]

4.2 Vývoj a testování

Na vývoji aplikace Výkaz práce se v současné době podílí 12 lidí (z toho 4 testeři). Vývoj je rozdělen mezi dva hlavní týmy. Tým testerů sídlí v Hradci Králové a tým vývojářů pracuje v pobočce v Brně.

Nová verze produktu bývá vydávána přibližně šestkrát za rok, a proto se musí během několika měsíců stihnout celý proces vývoje softwaru. Vývoj je řízený podle agilní metodiky, kde je rozdělen do několika iterací, v nichž mají jednotliví pracovníci přiřazeny úkoly s definovanou prioritou, které mají v dané iteraci vyřešit. Úkolem v tomto případě může být oprava nalezené chyby nebo naprogramování nové funkcionality.

Pro řízení vývoje softwaru a evidenci úkolů využíváme následující softwarové nástroje:

- **Jira** – Nástroj pro podporu řízení projektů a evidenci chyb a problémů.
- **Confluence** – Umožnuje vytvořit, organizovat práci v týmu a diskutovat o ní.
- **FishEye** – Zobrazuje poslední změny zdrojového kódu, commity a jiné aktivity. Veškeré aktivity v kódu jsou barevně zvýrazněny, což umožňuje jednoduchou kontrolu změn.
- **Crucible** – Umožňuje diskutovat jednotlivé řádky, soubory nebo celý seznam změn zdrojového kódu.
- **Xray** – Kompletní nástroj pro podporu řízení manuálního a automatického testování. [21]

Fáze testování může začít až v momentě, kdy jsou všechny úkoly v dané iteraci vyřešené a je zkontrolováno, zda obsahují všechny náležitosti. Zároveň musí být zveřejněna nová testovací verze aplikace.

Testování aplikace v první fázi probíhá na základě testovacích scénářů, které odpovídají vyřešeným úkolům. Testovací scénář obsahuje informace o prostředí, ve kterém test provádět, a popis, který obsahuje konkrétní postupy vystihující, jak otestovat vlastnosti a funkce aplikace. Na závěr testování se provádějí obecné testovací scénáře, které ověřují funkčnost všech částí systému, i těch, ve kterých se nic neměnilo, protože i do nich se mohla zanést chyba spojená s integrací nových funkcí. Všechny testovací scénáře musí být provedeny minimálně třemi testery, aby se zachytilo co možná nejvíce chyb.

Po dokončení verze, která je připravena ke zveřejnění, se vývoj a testování uzavře. Pokud je vše v pořádku, je možné verzi oficiálně zveřejnit. Celý systém se kontroluje po zveřejnění ještě jednou, nyní se ovšem jedná pouze o základní funkcionalitu.

Před dokončením této závěrečné práce je zautomatizované pouze testování jednotek, o které se starají vývojáři.

5 Automatizace testování

Intenzivní vývoj aplikace Výkaz práce přináší s každou novou verzí nové funkce a drobná vylepšení. S rozšiřováním aplikace dochází k navýšení množství obecných testovacích scénářů a doby jejich provádění. Aktuálně v systému evidujeme okolo padesáti obecných testovacích scénářů, které zahrnují celkovou kontrolu webové, desktopové i mobilní aplikace a jejich průměrná doba provádění je v řádech dnů. K testování dochází až na závěr, před zveřejnění verze, časově je možné je provést pouze jednou během procesu vývoje softwaru.

Jelikož manuální testování obsahuje řadu omezení, bylo dohodnuto s celým týmem zlepšit proces testování zavedením jeho automatizace, které povede ke zvýšení kvality výsledné aplikace a zrychlení procesu testování. Automatizované testy se spouští v každé iteraci ihned po zveřejnění funkční verze aplikace k testování, mohou být prováděny v několika různých prostředích zároveň, jsou vždy přesné a neúnavné.

V první fázi vývoje je automatizace testů zaměřena na uživatelské rozhraní webové aplikace, které je na testování nejnáročnější, obsahuje velké množství testovacích scénářů a testování bývá často monotónní a zdlouhavé. Tester může manuálním testováním ztratit koncentraci a jednotlivé kroky testovacího scénáře nemusí být po celou dobu testování prováděny přesně.

V dalších fázích zlepšení procesu testování se plánuje zavedení automatizace testování i pro mobilní a desktopovou aplikaci.

5.1 Analýza uživatelského rozhraní webové aplikace

Webová aplikace Výkaz práce je hlavní částí celého systému, definují se v ní veškerá nastavení. Správce systému má možnost definovat přístupové oprávnění pracovníkům, používané agendy nebo jejich rozšíření. Nastavení se sdílí mezi všemi klienty a zejména ovlivňuje uživatelské rozhraní všech aplikací.

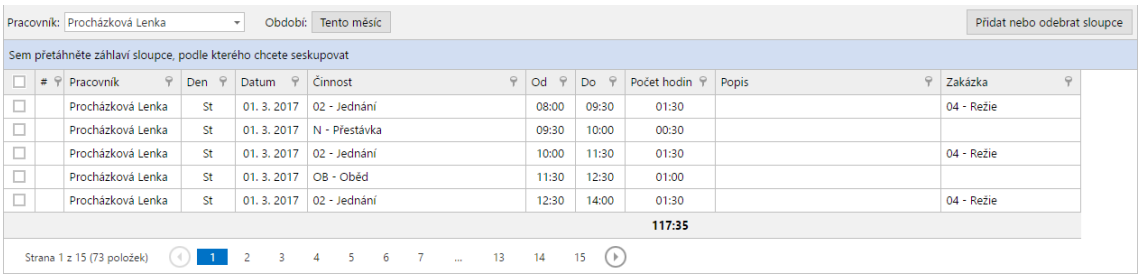
Jednoduché a přehledné uživatelské rozhraní webové aplikace je tvořeno několika základními ovládacími prvky, které se v rámci celé aplikace opakují. Každá stránka obsahuje panel nástrojů, formuláře pro založení a editaci záznamů a seznam

pro jejich zobrazení. V automatizovaných testech musíme brát v potaz, že celá aplikace je navíc lokalizována do tří jazyků a podporuje více platebních měn.

V následujících kapitolách budou podrobněji popsány jednotlivé části uživatelského rozhraní a ovládací prvky, které jsou nejvíce používané uživateli a jsou vhodné pro automatizaci.

5.1.1 ASPxGridView

ASPxGridView je nejpoužívanějším ovládacím prvkem z balíčku komponent DevExpress. V uživatelském rozhraní aplikace Výkaz práce sloužící k zobrazení jednotlivých záznamů v přehledné mřížce. Data v mřížce jsou zobrazena standardně v řádcích a sloupcích. Každý řádek reprezentuje jeden konkrétní uložený záznam a položky v jednotlivých sloupcích odpovídají atributům daného typu objektu. V případě většího počtu záznamů se v patičce mřížky aktivuje stránkování.



Obrázek 11: ASPxGridView
Zdroj: Vlastní zpracování

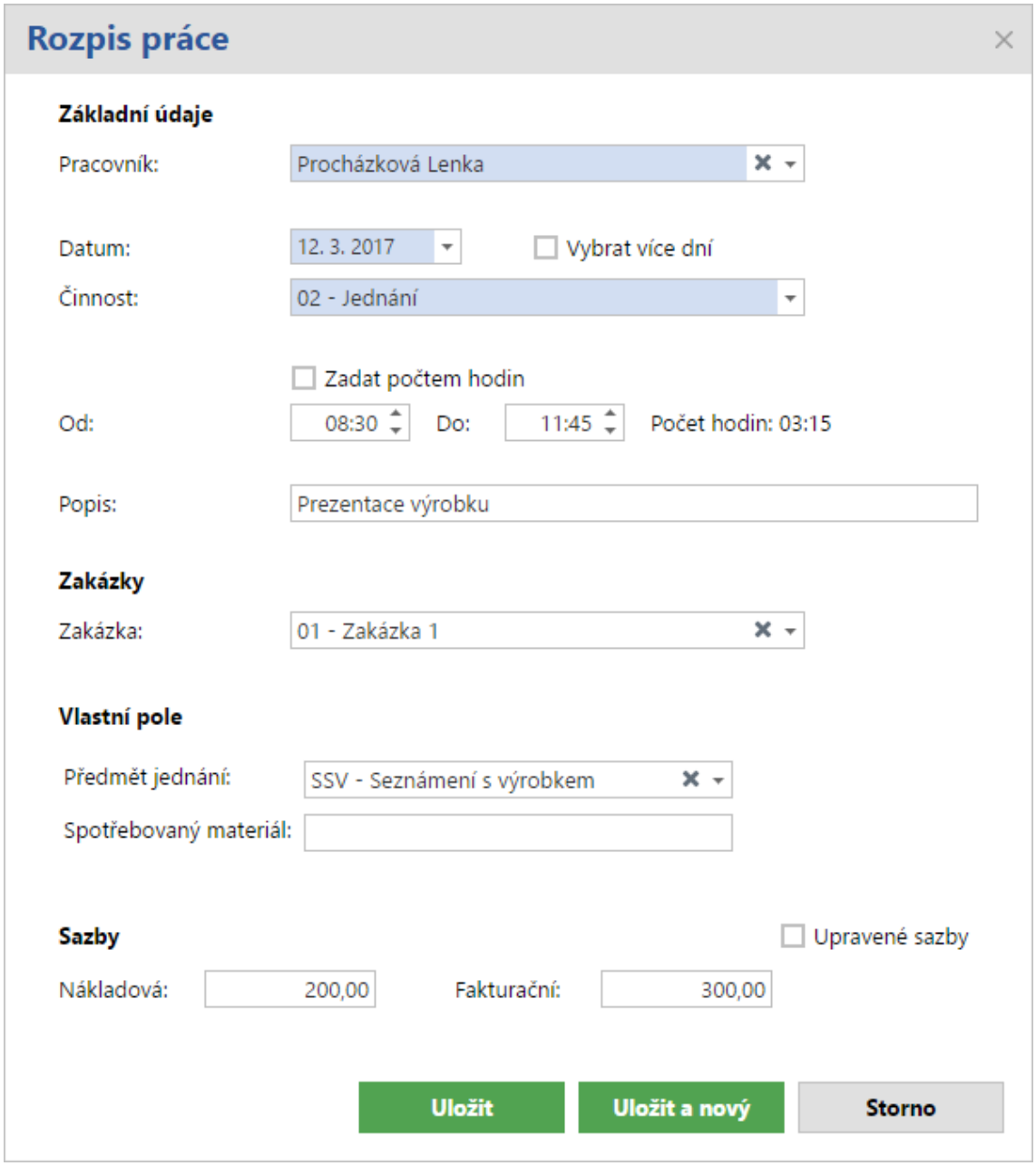
ASPxGridView nabízí mimo zobrazení dat mnoho dalších funkcí, které slouží ke konfiguraci mřížky nebo pro práci s daty. Data lze filtrovat, seskupovat nebo abecedně seřadit podle zvolených sloupců. V mřížce lze přidávat nebo odebírat jednotlivé sloupce, případně nastavit počet zobrazených záznamů na jedné stránce.

Uživatelsky velice příjemnou vlastností je zapamatování posledního nastavení mřížky. Nastavení je zachováno i po opuštění stránky a následném vrácení na stránku. To umožní, že data zůstanou seřazena stále stejně a uživatelé nemusí vytvářet pokaždé nové filtry.

Automatizované testy musí být schopny ověřit hodnoty položek v jednotlivých sloupcích, zda odpovídají hodnotám, které uživatel zadal v editačním formuláři. Dále musí umět ověřit zobrazení záznamů v mřížce na základě vybraného filtru.

5.1.2 Editační formuláře

Editační formuláře jsou neméně důležitou částí webového rozhraní a jsou určeny pro zakládání a editaci záznamů. Formuláře obsahují různé druhy ovládacích prvků. Mezi nejčastěji se vyskytující prvky patří tlačítka, vstupní pole, rozbalovací seznamy, zaškrtávací pole a komponenty pro výběr data a času. Uspořádaní těchto ovládacích prvků ve formulářích závisí na agendě, odkud jsou volány.

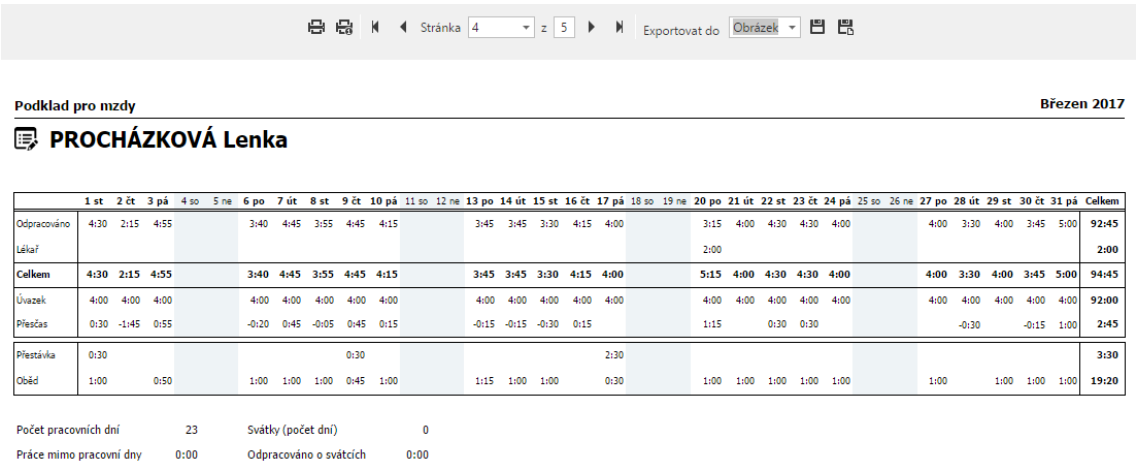


Obrázek 12: Editační formulář Rozpis práce
Zdroj: Vlastní zpracování

Špatně fungující formulář může negativně ovlivnit používání celé aplikace. Právě proto je pokrytí editačních formulářů automatizovanými testy velmi důležité. Automatizované testy jsou schopné za krátkou dobu projít všechny formuláře v aplikaci, kterých je okolo třiceti, a zkontrolovat jejich správné chování.

5.1.3 Tiskové sestavy a exporty

Systém nabízí přehledné předpřipravené tiskové sestavy a exporty. Jelikož některým uživatelům by předpřipravené dokumenty nevyhovovaly, je možné je uložit v několika formátech pro další zpracování.



	1 st	2 čt	3 pá	4 so	5 ne	6 po	7 út	8 st	9 čt	10 pá	11 so	12 ne	13 po	14 út	15 st	16 čt	17 pá	18 so	19 ne	20 po	21 út	22 st	23 čt	24 pá	25 so	26 ne	27 po	28 út	29 st	30 čt	31 pá	Celkem
Odpracováno	4:30	2:15	4:55			3:40	4:45	3:55	4:45	4:15			3:45	3:45	3:30	4:15	4:00			3:15	4:00	4:30	4:30	4:00			4:00	3:30	4:00	3:45	5:00	**92:45**
Lékař																				2:00												**2:00**
Celkem	**4:30**	**2:15**	**4:55**			**3:40**	**4:45**	**3:55**	**4:45**	**4:15**			**3:45**	**3:45**	**3:30**	**4:15**	**4:00**			**5:15**	**4:00**	**4:30**	**4:30**	**4:00**			**4:00**	**3:30**	**4:00**	**3:45**	**5:00**	**94:45**
Úvazek	4:00	4:00	4:00			4:00	4:00	4:00	4:00	4:00			4:00	4:00	4:00	4:00	4:00			4:00	4:00	4:00	4:00	4:00			4:00	4:00	4:00	4:00	4:00	**92:00**
Přesčas	0:30	-1:45	0:55			-0:20	0:45	-0:05	0:45	0:15			-0:15	-0:15	-0:30	0:15				1:15		0:30	0:30					-0:30		-0:15	1:00	**2:45**
Přestávka	0:30								0:30								2:30															**3:30**
Oběd	1:00		0:50			1:00	1:00	1:00	0:45	1:00			1:15	1:00	1:00		0:30			1:00	1:00	1:00	1:00	1:00			1:00		1:00	1:00	1:00	**19:20**

Počet pracovních dní	23	Svátky (počet dní)	0
Práce mimo pracovní dny	0:00	Odpracováno o svátcích	0:00

Obrázek 13: Tisková sestava Podklad pro mzdy
Zdroj: Vlastní zpracování

Ve starších verzích docházelo ke špatnému zobrazení tiskových sestav. Abychom tyto chyby minimalizovali, ukládáme snímky jednotlivých stránek, které pak manuálně procházejí testeři.

5.1.4 Ostatní

Mimo výše popsaných částí uživatelského rozhraní vhodných pro automatizaci, lze automatizovanými testy odchytávat i obecné chyby. Může se jednat o prvek, který na stránce nemá být zobrazen, nebo o chyby Javascriptu, které se zobrazují v konzoli webového prohlížeče.

5.2 Výběr testovacích nástrojů

Správný, a především včasný výběr vhodných testovacích nástrojů, je důležitou úlohou, na které závisí úspěšnost celé automatizace. Výběr testovacích nástrojů nelze podcenit, proto všechna kritéria byla po celou dobu konzultována společně s vývojáři. U každého uvažovaného nástroje byl proveden pilotní projekt k ověření, že nástroj plně podporuje naši využívanou technologii. Tímto krokem jsme snížili riziko, že během návrhu testů bude zjištěno, že některý ovládací prvek uživatelského rozhraní není podporován.

Výběr testovacích nástrojů probíhal podle následujících vybraných kritérií, seřazených od největší priority:

1. Cena a licenční podmínky
2. Správa testovacích případů
3. Udržovatelnost
4. Dostupnost a šíře podpory od výrobce nástroje
5. Cílový programovací jazyk C# a integrace do Visual studia
6. Analýza výsledků

Po zvážení všech kritérií bylo vybráno řešení založené na Selenium frameworku, za jehož hlavní výhody lze považovat:

- Open-source nástroj
- Množství tříd a funkcí pro práci s HTML elementy
- Přehlednou dokumentaci a velkou komunitu uživatelů
- Pravidelné vydávání nových verzí
- Psaní testovacích skriptů v jazyce C#
- Selenium framework společně s nástrojem Appium tvoří řešení pro testování mobilních aplikací
- Provádění testů na více počítačích

Naopak pokročilá znalost programování a složitější implementace testovacích skriptů mohou být vnímány za nevýhody.

Poslední nástroj, který byl potřeba vybrat, je framework pro jednotkové testování. V rámci konzultace s vývojáři byl vybrán NUnit, který právě vývojáři používají k testování jednotek.

Pro testování webové aplikace bylo tedy vybrané řešení zahrnující Selenium spolu s NUnit. Oceňujeme také řešení Selenium spolu s Appium pro budoucí testování mobilních aplikací, a to především z důvodu využití již získaných poznatků.

5.3 Testovací případy vhodné pro automatizaci

Ještě před tím, než začneme s implementací automatizovaných testovacích případů, je důležité se zamyslet, jaké testy budeme automatizovat, jelikož ne všechny testovací scénáře jsou vhodné k automatizaci.

Při výběru testů k automatizaci bychom měli zvážit následující kritéria [2]:

- **Stabilita** – Testovací případy, u kterých dochází k častým změnám, nemá příliš smysl automatizovat.
- **Četnost provádění** – Je vhodné automatizovat testy prováděné často a pravidelně.
- **Důležitost** – Testy ověřující kritickou funkcionalitu systému jsou prováděny na každém novém buildu.
- **Obtížné provádění** – O automatizaci uvažujeme, pokud se jedná o test, který je manuálně obtížně proveditelný. Jde o testy, které obsahují komplexní výpočty, nebo velké množství kroků.
- **Časová náročnost** – Určitý test vyžaduje značné množství času.
- **Složitost automatizace** – Některé testy lze automatizovat snadno (ověření hodnoty), jiné nikoliv (správné zobrazení grafiky).

V rámci naší webové aplikace Výkaz práce, byly vybrány k automatizaci nejdůležitější testovací scénáře, které je nutné provádět pokaždé v novém sestavení. Pokud by aplikace obsahovala chybu v nejdůležitějších testovacích scénářích a tester ji přehlédl, ovlivnilo by to negativně fungování celé aplikace, což by vedlo k negativním reakcím uživatelů, a zároveň k rostoucím nákladům na opravu chyby.

Právě kvůli snížení rizika vzniku těchto situací jsme se rozhodli pokrýt automatizovanými testy následující funkcionality:

- Registrace a přihlášení
- Založení, editace a mazání záznamů
- Exporty a tiskové sestavy
- Kontrola základních součtů

5.4 Implementace automatizovaných testů

Samotná implementace automatizovaných testů pro webovou aplikaci, jak již bylo zmíněno, je postavena na frameworcích NUnit a Selenium. Celý projekt je napsán v jazyce C# a pro psaní, spouštění a ladění testů se využívá vývojové prostředí Visual Studio.

5.4.1 Struktura projektu

Projekt je již v době psaní této závěrečné práce velmi rozsáhlý a obsahuje velký počet testovacích tříd a metod, které napomáhají tomu, aby automatizované testy byly snadno udržovatelné.

V následujících kapitolách budou popsány základní skupiny tříd, které jsou nejdůležitější pro fungování celého projektu.

5.4.1.1 Inicializační třídy

Inicializační třídy, zpravidla označeny atributem SetUpFixture, můžeme rozdělit na třídy sloužící k inicializaci prostředí a na třídy k inicializaci aplikace.

Třídy k inicializaci prostředí slouží především k nastavení prostředí, ve kterém mají testy probíhat. Třídy obsahují parametry, které umožňují zvolit a konfigurovat webový prohlížeč podle potřeby, nebo definovat, zda testy mají být spuštěny na lokálním či vzdáleném počítači.

Třídy k inicializaci aplikace slouží k nastavení webové aplikace. Tyto třídy se starají o registraci účtu, přihlášení k účtu a založení základních objektů, které testy vyžadují pro svůj běh.

5.4.1.2 Testovací třídy

Každá testovací třída označena TestFixture odpovídá jedné webové stránce v aplikaci. Třída obsahuje testovací metody a pomocné metody, které se starají o vyplnění formuláře nebo vyhledávaní uložených záznamů v seznamu.

Testovací metody obsahují sekvenci kroků, které vedou k ověření konkrétní funkcionality. V prvním kroku dojde k přesměrovaní pomocí url adresy do dané agendy, kde se budou provádět následující kroky testu.

5.4.1.3 FormData třídy

Ke každé testovací třídě je přiřazena FormData třída, která definuje data pro testovací metody. Třída FormData obsahuje atributy, které odpovídají datovým typům formuláře, a konstruktory, které vytvářejí specifické sety dat. Data jsou předána testovacím metodám pomocí atributu TestCaseSource. Výhodami těchto tříd je znuvupoužitelnost a oddělení inicializace dat od testů, což vede k přehlednosti testovacích metod.

5.4.1.4 Pomocné třídy

Poslední skupinou jsou pomocné třídy. Tyto třídy obsahují metody pro vyhledávání ovládacích prvků a práci s nimi. Vyhledávání konkrétních elementů je většinou implementované podle identifikátoru, css selektoru nebo pomocí jazyka XPath.

Důležitou vlastností, která se využívá právě v metodách pro vyhledávání ovládacích prvků na webové stránce, je explicitní čekání. Explicitní čekání je kód, který definuje dobu, kterou se má vyčkat, než je splněna určitá podmínka před pokračováním dále v kódu. Pokud by vyhledání neobsahovalo toto čekání, test by pravděpodobně skončil chybou a dostali bychom výjimku NoSuchElementException, neboť element v době hledání ještě neexistoval.

```
public static void ClickElementById(string idToFind)
{
        WebDriverWait wait = new WebDriverWait(Session.Driver,
TimeSpan.FromSeconds(10));

        IWebElement element =
wait.Until(ExpectedConditions.ElementToBeClickable(By.CssSelector($"[id$='{i
dToFind}']")));
        element.Click();
}
```

Obrázek 14: Ukázka explicitního čekání
Zdroj: Vlastní zpracování

V ukázce výše čeká WebDriverWait až deset sekund, než vyhodí výjimku TimeoutException. Standardně se dotazuje na element, zda je klikatelný, každých 500 ms. Pokud ExpectedConditions vrátí hodnotu true, je vykonán klik na element.

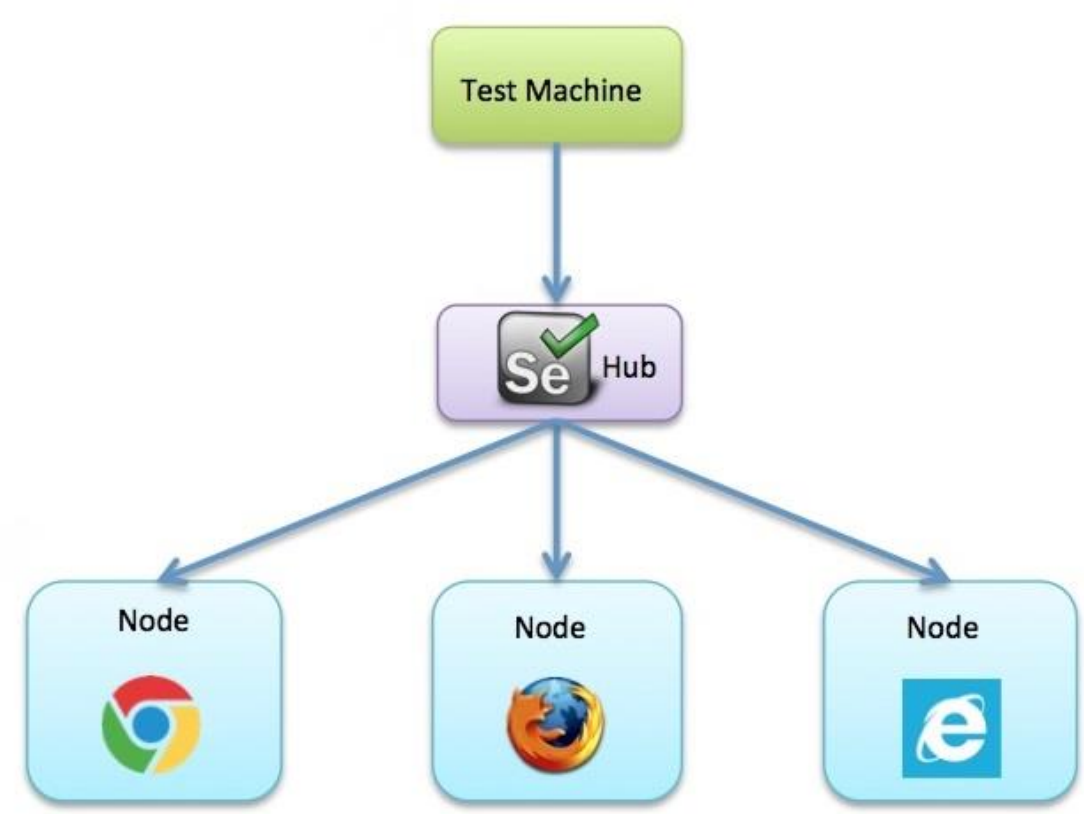
Alternativy k explicitnímu čekání jsou Implicitní čekání a Thread.Sleep(). Tato řešení jsou zastaralá a nejsou aktuálně doporučovaná, protože výrazně prodlužují běh testů.

Další pomocné třídy slouží pro práci s mřížkou nebo k uložení všech unikátních identifikátorů elementů jako konstanty, aby v budoucnu byly jednoduše udržovatelné.

5.5 Spouštění automatizovaných testů

Spuštění testů nad webovými prohlížeči je možné realizovat různými možnostmi. Nejběžnější je spouštění na lokálním počítači. Výhodou tohoto způsobu je rychlost a jednoduchá počáteční konfigurace. Nevýhodou je, že testy nelze spouštět v odlišných verzích webového prohlížeče na různých operačních systémech.

Tyto nedostatky řeší Selenium Grid, nástroj, který spouští testy v různých webových prohlížečích na různých operačních systémech na vzdálených zařízeních. Nevýhodou je zejména složitější počáteční konfigurace, kdy je potřeba si všechny vzdálené počítače nastavit. Konfigurace zahrnuje instalaci operačního systému, instalace prohlížečů, nastavení hubu a vytvoření nodu na vzdálených počítačích.



Obrázek 15: Selenium Grid 2.0
Zdroj: Vlastní zpracování

Konfigurace různých vzdálených počítačů lze snadno outsourcovat a využít služeb některých cloudových poskytovatelů. Řešení nabízí stejné služby jako Selenium Grid, navíc u nich odpadá nutnost konfigurace vzdálených počítačů. Připojení ke službě je realizováno částí kódu od poskytovatele doplněnou o vygenerovaný přístupový klíč. Nevýhodou je, že tato služba je zpoplatněna. V našem projektu jsme využili zkušební verzi nabízenou firmou TestingBot.

5.6 Vyhodnocení automatizovaných testů

Po provedení všech testů nabízí Visual Studio v Test exploreru přehledné informace o stavu jednotlivých testů. Test explorer seskupuje testy do tří skupin podle jejich výsledků na:

- **Passed** – Testy proběhly v pořádku.
- **Failed** – Testy obsahují chybu. U testů se zobrazí výpis, v kterém kroku selhaly. V projektu jsme dodatečně implementovali funkci, která vytvoří screenshot obrazovky v případě, že test selže. Umožňuje to i visuální kontrolu, proč došlo k selhání testu.
- **Warning** – Testy s výstražnou zprávu. Výstražná zpráva je zobrazena, pokud není splněna některá z podmínek.
- **Skipped** – Testy, které byly přeskočeny. Testy mohou být přeskočeny z důvodu nefunkčního testu, který čeká na opravu.

Vyhodnocení testů provádíme na základě získaných informací o výsledcích testů a na základě kontroly snímků obrazovky. Snímky obrazovky využíváme pro kontrolu grafického rozhraní webové aplikace. Každý vyplněný editační formulář je zachycen tak, jak je zobrazen. Tester následně manuálně prochází veškeré uložené snímky spolu s exporty a ověřuje, že vše je zobrazeno správně.

6 Závěr

Cílem bakalářské práce bylo seznámit čtenáře srozumitelnou formou s testováním softwaru, zařadit testování do životního cyklu vývoje softwaru, popsat možnosti automatizovaného testování, včetně výběru vhodných nástrojů, a na základě získaných znalostí navrhnout a implementovat zlepšení procesu ve společnosti Systemart s. r. o.

V první části bakalářské práce byly uvedeny základní informace o testování softwaru. Následující kapitoly popisují způsoby testování podle toho, v jaké úrovni a s jakým časovým odstupem od napsání zdrojového kódu se testování provádí, nebo jak k testované aplikaci přistupují.

Kapitola o modelech životního cyklu softwaru se věnuje základním modelům, které využívají společnosti pro projektové řízení. Popisuje model vodopádu, spirálový model, RUP a na závěr metodiky agilního vývoje, které jsou v současné době velmi využívané. U každého modelu je shrnutí výhod a nevýhod z hlediska testování.

Na základě informací z teoretické části a známých chyb z historie lze dojít k závěru, že testování patří do každého životního cyklu softwaru a nelze tuto část vynechat, nebo jí věnovat méně pozornosti. Náklady na odstranění chyb ve zveřejněném softwaru mohou překročit úspory vzniklé vynecháním testů.

Významná část bakalářské práce je věnována nástrojům pro automatizaci testování, včetně shrnutí jejich výhod a nevýhod. Jsou zde představeny frameworky pro jednotkové testování a nástroje pro automatizované testovaní webových, mobilních a desktopových aplikací.

Závěr práce je věnován popisu průběhu procesu testování ve společnosti Systemart s. r. o., analýze aplikace Výkaz práce, výběru vhodných nástrojů a implementaci výsledného řešení. Pro implementaci výsledných automatizovaných testů byl vybrán nástroj Selenium framework s kombinací s NUnit frameworkem na jednotkové testování.

Hlavní přínos své práce vidím v zavedení automatizovaného testování webové aplikace, kde jsou vypracované testy používány při vývoji nových verzí a staly se nedílnou součásti vývojového cyklu. Automatizované testy jsou v každé

nové verzi aplikace udržovány, rozšiřovány a pomohly již k nalezení mnoha chyb. Ve společnosti se počítá i se zavedením automatizovaného testování pro mobilní a desktopové aplikace.

Je důležité si uvědomit, že zvolené postupy a nástroje byly vybrány pro konkrétní společnost a aplikaci. V případě jiné společnosti, nebo aplikace, může dojít k výběru jiných, vhodnějších nástrojů a přístupů.

7 Seznam použité literatury

[1] **PATTON, Ron**. *Testování softwaru*. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.

[2] **ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ**. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

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

[4] **BALAJI, S.; MURUGAIYAN, M. Sundararajan**. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2012, 2.1: 26-30.

[5] **HLAVA, Tomáš**. Testování softwaru. Testování softwaru [online]. 2011 [cit. 2017-01-25]. Dostupné z: http://www.testovanisoftwaru.cz

[6] *Rational Unified Process: Best Practices for Software Development Teams* [online]. Rational Software [cit. 2017-01-25]. Dostupné z: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

[7] **STEPHENS, Matt a Doug ROSENBERG**. *Testování softwaru řízené návrhem*. Brno: Computer Press, 2011. ISBN 978-80-251-3607-2.

[8] **HUNT, Andrew a David THOMAS**. *Programátor pragmatik: jak se stát lepším programátorem a vytvářet kvalitní software*. Brno: Computer Press, 2007. ISBN 978-80-251-1660-9.

[9] **OSHEROVE, Roy**. *The art of unit testing: with examples in .NET*. Greenwich, CT.: Manning, 2009. ISBN 1933988274.

[10] **JUnit – About**. *JUnit* [online]. 2017 [cit. 2017-01-28]. Dostupné z: http://junit.org/junit4/

[11] **NUnit.org**. *NUnit* [online]. [cit. 2016-11-10]. Dostupné z: http://www.nunit.org

[12] **NUnit Documentation**. *How people build software* [online]. 2016 [cit. 2017-01-28]. Dostupné z: https://github.com/nunit/docs/wiki/NUnit-Documentation

[13] **XUnit.net**. *XUnit.net* [online]. 2016 [cit. 2017-01-25]. Dostupné z: http://www.xunit.github.io

[14] **Unit Test Basics**. *Learn to Develop with Microsoft Developer Network | MSDN* [online]. 2016 [cit. 2017-01-25]. Dostupné z: https://www.msdn.microsoft.com/en-us/library/hh694602.aspx

[15] **Automating User Interface Tests**. *Android Developers* [online]. [cit. 2017-01-25]. Dostupné z: https://www.developer.android.com/training/testing/ui-testing/index.html

[16] **Selenium - Web Browser Automation**. *Selenium - Web Browser Automation* [online]. [cit. 2017-01-28]. Dostupné z: http://www.seleniumhq.org

[17] **WatiN**. *WatiN* [online]. [cit. 2017-01-28]. Dostupné z: http://www.watin.org

[18] **Appium: Mobile App Automation Made Awesome**. *Appium: Mobile App Automation Made Awesome.* [online]. [cit. 2017-01-28]. Dostupné z: http://www.appium.io

[19] **Calaba.sh - Automated Acceptance Testing for iOS and Android Apps**. *Calaba.sh - Automated Acceptance Testing for iOS and Android Apps* [online]. [cit. 2017-01-28]. Dostupné z: http://www.calaba.sh

[20] **Automated Software Testing**. *Software Testing, Monitoring, Developer Tools | SmartBear* [online]. 2017 [cit. 2017-01-28]. Dostupné z: https://www.smartbear.com/product/testcomplete/overview

[21] **Atlassian JIRA** [online]. Onlio, c2016 [cit. 2017-03-25]. Dostupné z: http://www.myjira.cz

[22] **Výkaz práce** [online]. Systemart, c2003-2017 [cit. 2017-03-25]. Dostupné z: http://www.vykazprace.cz

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Burda Tomáš	Na Vinici 313, Vysoké Veselí	I14066

TÉMA ČESKY:

Testování softwaru

TÉMA ANGLICKY:

Software testing

VEDOUCÍ PRÁCE:

doc. RNDr. Petra Poulová, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

1. Úvod
2. Teoretická část
   a. Testování softwaru
   b. Nástroje pro testování
      i. Frameworky pro jednotkové testování
      ii. Nástroje pro automatizaci testování UI
3. Praktická část
   a. Aplikace Výkaz práce
      i. Popis aplikace
      ii. Vývoj a testování
   b. Automatizace testování
      i. Analýza uživatelského rozhraní webové aplikace
      ii. Výběr testovacích nástrojů
      iii. Implementace automatizovaných testů
      iv. Spouštění automatizovaných testů
      v. Vyhodnocení automatizovaných testů
4. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.

Podpis studenta: ..... **Datum:** 20.4.2017

Podpis vedoucího práce: ..... **Datum:** 20.4.2017