TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Podpora vývoje v průběhu životního cyklu aplikace

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Lukáš Vosecký**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Application support of development through its life-time

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Lukáš Vosecký**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2017/2018**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš Vosecký**
Osobní číslo:	**M15000060**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Podpora vývoje v průběhu životního cyklu aplikace**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s požadavky na ladění, údržbu, monitoring a deployment .NET aplikací na běžných platformách.
2. Implementujte vybrané techniky s využitím vhodně vybraných nástrojů.
3. Na vlastní ukázkové aplikaci demonstrujte správnou a spolehlivou funkci implementovaných technik.
4. V závěru shrňte dosažené výsledky a diskutujte možnosti dalšího rozvoje tématu.

Rozsah grafických prací:	**dle potřeby dokumentace**
Rozsah pracovní zprávy:	**30–40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Deploying the .NET Framework and Applications: .NET Framework 4.6 and 4.5 [online]. [cit. 2015-10-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/6hbb4k3e(v=vs.110).aspx**

[2] **AppDynamics App iQ Platform Documentation [online], 4.3. [cit. 2017-10-13]. Dostupné z: https://docs.appdynamics.com/**

[3] **GOLDSHTEIN, Sasha, Dima ZURBALEV a Ido FLATOW, 2012. Pro .NET Performance: Optimize Your C# Applications. Apress. ISBN 1430244585.**

Vedoucí bakalářské práce:	**Ing. Jan Kraus, Ph.D.**
	Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce:	**10. října 2017**
Termín odevzdání bakalářské práce:	**14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

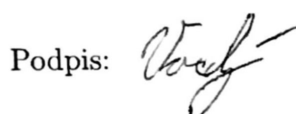
Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2018

Podpis: 

Poděkování

Poděkování patří především panu Ing. Janu Krausovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při zhotovování bakalářské práce.

// Abstrakt

Instalace a aktualizace aplikací je v nynější době brána za velkou problematiku v oblasti deploymentu. Do zároveň zmiňované problematiky zapadá i monitoring, ladění a údržba. Tyto segmenty deploymentu není vždy nejlehčí zpracovávat.

Bakalářská práce se zaměřuje právě na oblast deploymentu. V práci jsou popsány možnosti instalace a aktualizace .NET aplikací, kde největší důraz je kladen na celkovou automatizaci vytváření instalačních souborů. Zároveň je kladen důraz na tvoření tichých instalací. Dále práce obsahuje popis metodik pro monitoring, ladění a údržbu. Nakonec ze všech těchto technik jsou vybrané nejlepší nástroje a ty jsou aplikovány na ukázkové aplikaci.

Klíčová slova:

.NET, instalace a aktualizace .NET aplikací, životní cyklus aplikace, logging, monitoring, ladění a údržba .NET aplikací

Abstract

Installation and actualization is nowadays one of the biggest problematic issue in the field of deployment. At the same time it includes monitoring, tuning and maintenance. These segments of deployment aren't the easiest to process.

This work focuses on deployment and takes a closer look at the possibilities of installation and actualization of .NET applications, where the biggest emphasis is laid on the full automatization of installation folders architecture including silent installations. This work also contains methodical strategies for monitoring, tuning and maintenance. In the end of all these techniques is a selection of special tools that are applied in an example application.

Key words:

Installation and actualization of .NET applications, application life cycle, logging, monitoring, tuning and maintenance of .NET applications

Obsah

Seznam zkratek . . . 11

1 Úvod 12

2 Životní cyklus vývoje aplikací 13

2.1 Traditional Software Development . . . 13

2.1.1 Vývojové fáze - model Waterfall . . . 13

2.1.2 V-Model . . . 16

2.2 AGILE Software Development . . . 16

2.2.1 Extreme Programming . . . 17

2.2.2 Scrum . . . 17

2.2.3 Feature Driven Development . . . 17

2.3 Vyhodnocení Tradiční a Agilní metodiky . . . 18

2.4 Odchytávání chyb . . . 18

2.5 Diagramy . . . 19

2.6 Logging . . . 20

3 Instalace a aktualizace aplikací 22

3.1 Podpisy kódu . . . 22

3.1.1 Účel podpisů . . . 23

3.1.2 Certifikáty na podpisy kódů . . . 23

3.1.3 Důvěra podpisů . . . 23

3.1.4 Jak to funguje . . . 23

3.2 Instalace aplikací . . . 24

3.2.1 Problematika instalace aplikací . . . 24

3.2.2 Způsoby instalace . . . 25

3.3 Aktualizace aplikací . . . 26

3.4 Microsoft Visual Studio 2017 Installer Projects . . . 27

3.4.1 Výsledek po testování . . . 27

3.5 Advanced Installer . . . 28

3.5.1 Výsledek po testování . . . 28

3.6 WiX Toolset . . . 28

3.7 Nástroj msbuild . . . 31

3.8 Vyhodnocení nástrojů pro deployment . . . 32

4 Monitoring aplikací **33**
4.1 Ladění a údržba 34
4.2 Vybrané nástroje pro monitoring, ladění a údržbu 34
4.2.1 Ukázkové aplikace a zkoušené vlastnosti nástrojů 35
4.2.2 Log4net 35
4.2.3 NLog 41
4.3 Vyhodnocení nástrojů pro monitoring 42

5 Výsledné zhodnocení testováných nástrojů **43**
5.1 Deployment 43
5.2 Monitoring 43
5.2.1 Cloudové řešení pro log4net 44

6 Závěr **45**

A Obsah přiloženého CD **49**

Seznam obrázků

4.1 Znázornění grafu pro výpis chyb za den 38
4.2 Výpis nejčastějších a nedávno logovaných chyb 39
4.3 Příchozí e-mail po nově vzniklé chybě 39
4.4 Každodenní výpis logu 40

Seznam tabulek

5.1 Hodnocení nástrojů pro deployment na základě testování 43
5.2 Hodnocení nástrojů pro monitoring na základě testování 44
5.3 Hodnocení cloudových řešení pro log4net 44

Seznam zkratek

SDLC Software development life cycle
XP Extreme Programming
UML Modelovací jazyk
ERD Vztahový model
DFD Diagram datových toků
FDD Feature Driven Development
msi Formát instalačního souboru
exe Spustitelný soubor
GUI Grafické uživatelské rozhraní
XML Značkovací jazyk

1 Úvod

Práce vznikla z důvodu častého využití v malých i velkých projektech a malé diskutovanosti tématu.

Práce pojednává o celkovém vývoji životního cyklu aplikace, kde jsou popsány jednotlivé části vývoje v různých metodikách. Zároveň v celém životním cyklu je dále kladen největší důraz na segment deploymentu (zavedení aplikace do prostředí).

Postupně je popsána problematika jednotlivých fází životního cyklu a následně problematika deploymentu. Problematika jednotlivých fází je popisována z důvodu fáze návrhu, která je celkově nejdůležitější fází v celém vývoji systému. Pokud je aplikace dobře navržena, je poté lehce monitorovatelná. Tím pádem se snáze hledají a opravují chyby. Ve fázi deploymentu je kladen nejvíce důraz na instalaci .NET aplikací. Z počátku jsou diskutovány různé možnosti vytváření instalačních souborů a ke konci je v práci pojednáváno už s konkrétními nástroji, které řeší potřebnou problematiku. Touto problematikou je myšleno vytváření instalačních souborů pomocí skriptů (celková automatizace). Následuje instalace na koncovém počítači pomocí tichých instalací. Vybranými nástroji jsou WiX Toolset
a msbuild, které jsou testovány na aplikaci vytvořené k účelům této práce a následně jsou zhodnoceny. Zároveň jsou testovány instalace prerekvizit jak aplikací třetích stran, tak i například .NET Framework.

V předposlední fázi jsou diskutovány nástroje pro monitoring, kde je popsána celková problematika a nakonec samotné testování vybraných nástrojů (log4net, NLog). Dalším krokem je nasazení těchto nástrojů na ukázkovou aplikaci a nakonec jejich zhodnocení.

V závěru jsou shrnuty dosažené výsledky z celkového testování, zároveň z implementování technik na vytvořenou aplikaci a diskutování o možnosti dalšího rozvoje tématu.

2 Životní cyklus vývoje aplikací

Životní cyklus vývoje aplikací (dále SDLC) je proces vytváření software a jeho udržování. Vývoj aplikací se v dnešní době šíří mnoha směry a zároveň mnoha způsoby. V této práci se zabývám vývojem `C#` aplikací ve vývojovém prostředí Visual Studio. Vývojových prostředí je samozřejmě mnoho, ale pro můj účel je VS nejlepší možností, jelikož se zajímám o `C#` aplikace. Aplikace `C#` se dají vyvíjet i v jiných vývojových prostředích, než ve Visual Studiu, ale ve škole jsem s tímto vývojovým prostředím setkával poměrně často, právě proto jsem zvolil toto vývojové prostředí. Kdybych se ale zajímal například o Java aplikace, zvolil bych prostředí NetBeans. Další vhodné prostředí je například Eclipse.

Ve většině případech SDLC zahrnuje více rozdílných fází od základního zjišťování informací až po samotný vývoj softwaru a testování. Ačkoliv existuje mnoho druhů metodik, v této práci se zaměřím na dvě metodiky, které jsou zároveň nejčastěji používané ve světě. Jsou jimi **Traditional Software Development a AGILE Software Development**[1]. Každá z těchto metodik potom využívá rozdílné metodiky, které se používají k vývoji, jak bude popsáno v dalších kapitolách.

2.1 Traditional Software Development

První a zároveň nejvíce používanou metodikou je Tradional Software Development. Je to metodika založená na několika za sebou jdoucích fázích, které samozřejmě mohou být podle potřeby různě upraveny, ale princip a pořadí musí být zachovány. Zároveň se fáze mohou lišit podle dalších metodik, které pod tuto zastřešující metodiku spadají. Jejími příklady jsou Waterfall a V-model [1], které budou následně popsány. Nejtradičnější začátek ve většině fázích je na úvod zjištění všech požadavků od zákazníka, kde se odhaduje délka trvání celého procesu a zároveň se shromažďují všechny potřebné informace. Dále následuje fáze návrhu, kde se navrhuje celý systém. Následně nastává fáze kodingu a testování. Poté už jen zbývá deployment.

2.1.1 Vývojové fáze - model Waterfall

Waterfall je klasickým modelem softwarového inženýrství. Patří mezi jedny z nejstarších modelů a je velmi užívaný ve velkých a vládních korporacích. Mnoho lidí věří, že tato metoda není použitelná pro všechny situace. Například, pokud vezmeme čistý Waterfall model, všechny požadavky musí být dány a vyhodnoceny před návrhem

a návrh musí být uskutečněn před vývojem. Neexistuje žádné překrývání fází. Nic méně ve skutečném světě vývoje, vývojář může narazit na problémy při codingu, které ukazují na chyby nebo mezery ve fázi návrhu nebo shromažďování informací. Metoda ovšem nezakazuje vracet se o fázi nebo fáze zpět, ačkoliv to může znamenat předělání celé fáze a právě fáze následující. A protože skutečný vývoj přichází až po návrhu, může dojít k velkému zdržení [2].

Na samotném začátku je potřeba znát přesné požadavky na aplikaci, tedy co aplikace musí umět, na jaké platformě má být spuštěná (nebo na více platformách) a pro jakou skupinu uživatelů je aplikace určená. Této fázi se říká **Analýza požadavků zákazníka**. Je to ta nejkritičtější fáze, protože pokud v této fázi dojde k nějakému nedorozumění, tak výsledkem je často chybná aplikace. Tedy team vývojářů (nebo vývojář) často komunikuje se zákazníkem pomocí jednoduchých frekventovaných otázek, nebo pro urychlení je možnost vytvoření dotazníků, které pokryjí větší část dotazů. V praxi je to tak, že od zákazníka dostáváme velice zjednodušené zadání a programátoři poté zahlcují zákazníka dotazy k zhotovení aplikace.

Další fází je **Design**, kterou řeší zpravidla vedoucí vývoje. V této fázi je v podstatě diskutováno, jak bude aplikace psána, transakce nad aplikací, bezpečnost, hardwarové a systémové požadavky. Zde je tedy už zapotřebí vědět, na jakých platformách bude aplikace fungovat, právě protože by mohly nastat problémy se spouštěním aplikace na více platformách. Kritickou roli tu hraje také bezpečnost, jelikož se musí navrhnout celkové zabezpečení aplikace, což dále souvisí s legálními záležitostmi. Tím je myšleno, pokud do aplikace budou vstupovat uživatelé s citlivými daty, tak jak budou data ukládána a kam, dále autentizace nebo autorizace uživatelů. Toto je důležité řešit pro všechny společnosti.
Dále se řeší funkcionalita aplikace. Pokud budou v aplikaci nějaká pole pro vyplňování (text, čísla) mohou být pole vynechaná (prázdná)? Nebo musí být všechna vyplněna, popřípadě která ano a která ne. Jestli některá pole budou požadovat přesný formát, určitý počet písmen nebo čísel. Co se stane po kliknutím na nějaké tlačítko, která scéna se má nebo nemá ukázat? Zde se právě také řeší sledování chyb, jak bude popsáno dále.
Nakonec se zde řeší funkcionality, které né zcela souvisí s chodem aplikace. Takže například jak aktualizovat aplikaci, velikost kterou bude zabírat a jak to bude ovlivňovat systém. Rychlost aplikace a jak bude rychle komunikovat s uživatelem a také se serverem nebo databází.

Nyní následuje samotné psaní aplikace, nebo-li **Coding**. V této fázi vývojáři píši aplikaci podle požadavků, které jim předal hlavní vývojář. V této fázi je důležité aby vývojáři měli otevřenou mysl dalším nápadům a vylepšením, jelikož se může stát, že hlavní vývojář přijde s nějakou novinkou nebo vylepšením, které mu zákazník řekl. Proto se doporučuje psát kód tak, se mohl lehce předělat nebo rozšířit (tedy abstraktně). V této fázi se také píši takzvané unit testy. Jsou to testy, které vývojář píše tak, aby otestoval kousky (bloky) kódu samostatně. Právě díky unit testování je snazší upravit kód tak, aby fungoval stále stejně. Dochází zde k dokumentaci kódu,

která slouží pro lepší orientaci. Samotná dokumentace je velice důležitá, ačkoliv je velice zanedbávaná. Pokud nastane situace, kdy se ke kódu dostane jiný team, nebo celkově jiný člověk, samozřejmě se v něm hned nevyzná. A to neplatí pro jiného člověka, sám vývojář který píše aplikaci a například měsíc svůj kód neviděl, je celkem zřejmé, že si nebude pamatovat co kde napsal. Tedy nejlepší scénářem je dokumentovat každý blok kódu.
Zde je dobré po nějaké době, nebo po nějakém dokončeném úseku ukázat zákazníkovi jak produkt vypadá, aby ho on sám mohl ohodnotit. Zde se nejlépe ukáže, zda se vydáváte správným směrem nebo jestli je potřeba něco upravit. V podstatě je tato fáze tou nejdelší fází SDLC, jelikož je zde potřeba stála kontrola a dost často se kód předělává.

V předchozí části jsem již zmínil testování. To je právě hlavním účelem následující fáze, která právě nese stejný název: **Testing**. Jakmile je kód aplikace dopsán, nebo nějaká její část, přichází na řadu testování. Dost často v menších projektech je tato fáze zanedbávána, jelikož je to pracná fáze a často trvá delší dobu, jelikož je potřeba otestovat všechny možné situace. Právě zde se ukáže, jestli je zákazník spokojen s produktem, jestli aplikace nemá chyby, a celkově jestli je vše v pořádku. Pokud není, vrací se vývojáři o pár kroků zpět, a musí kód přepsat. Někdy se i stává, že se musí team vrátit k fázi Designu, kde je potřeba předělat návrh aplikace a poté opět změny putují k vývojářům. Je dobré zde testovat všechny krajní meze, které mohou nastat a nic nezanedbat, protože může potom nastat chyba v provozu, a to si nemůžeme dovolit. Jakmile je vše schváleno a odsouhlaseno přistoupíme k další fázi.

Následující fází je nasazení aplikace do prostředí - **Deployment**. Nasazením aplikace do prostředí je v podstatě myšleno vytvoření instalačního programu, který nainstaluje všechny potřebné komponenty na koncové zařízení. Jakým způsobem bude deployment prováděn je již určeno ve fázi plánovaní. Figuruje zde mnoho faktorů, které určují, jak bude aplikace distribuována. Hlavními faktory jsou zejména velikost aplikace, využití aplikace a jestli bude aplikace v budoucnu rozvíjena. Velikostí aplikace se myslí počet programů a různých souborů, která s sebou aplikace nese. Dále kdo a jak bude aplikaci používat, to je tedy využití aplikace a nakonec jestli bude zapotřebí aplikaci v budoucnu upravovat, opravovat nebo dále vyvíjet (updaty aplikace). Více o této fázi bude napsáno v dalších kapitolách, jelikož je to rozsáhlé téma 3.

Výhody metodiky Waterfall:

- Jednoduchý na pochopení a implementaci
- Široce používaný a známý
- Používá dobré zvyky: define-before-design, design-before-code

Nevýhody

- Spíše teoretický, nehodí se úplně do praxe

- Nerealistické očekávání všech přesných požadavků na začátku
- Vývoj je až v pozdní fázi, tedy zpoždění objevení chyb [2]

2.1.2 V-Model

Tento model je lehce odlišný od předchozího. A to ve smyslu testování. V-model klade velký důraz na testování, a to dokonce na konci každé fáze. Z toho tedy vyplývá, že má dokonce více testů. To má zároveň několik výhod, například zamezení selhání v jednotlivých fázích. Ovšem stejně jako u metody Waterfall, jsou fáze na sobě závislé a jdou postupně. Tedy k začátku nové fáze je nejprve potřeba zakončení té předchozí [2]. Samotné fáze jsou lehce odlišené, zejména jak jsem již uvedl kvůli testování, které se provádí na konci každé fáze, ale i v jejich základu. První fází je opět shromažďování požadavků, kde zákazník sděluje všechny náležitosti. Následují dvě fáze návrhu. V první fázi návrhu je důrazně zaměření na samostatnou architekturu systému. Kdežto ve druhé fázi návrhu je kladen důraz spíše na samotné softwarové náležitosti [2]. Nakonec je samotná implementace, tedy psaní kódu.

Výhody metodiky V-Model:

- Jednoduchý na pochopení a použití
- Větší šance na úspěšnost díky testům v předešlých fázích
- Dobré použití pro menší projekty

Nevýhody

- Software se provádí až ve fázi implementace, takže není možnost brzkých prototypů
- Neprovádí jasné řešení při nalezení problému v testovacích fází [2]

2.2 AGILE Software Development

Začněme zavedením si slova Agile. Slovo agile znamená hbitý, čilý. První co nás napadne, je tedy více flexibilní metodika. V Traditional Software Development jsme měli dané fáze, které na sebe navazovaly a bylo velice obtížné a náročné se jakkoliv vracet zpět. V této metodice je to přesně naopak. Jsou opět dané fáze, ale celé řešení systému je nastaveno s větší flexibilitou a lepší možností vracet se zpět. Hlavním důvodem je rychlost celého procesu a zákaznická spokojenost, jelikož zákazník je schopen vidět prototyp dříve, než mu je dodána finální verze. Je zde hodně kladen důraz na komunikaci se zákazníkem, a to při běhu celého procesu vývoje software[3].

2.2.1 Extreme Programming

Extreme Programming je první vybranou metodikou z Agile Software Development. Je to poněkud zvláštní postup, jak už název naznačuje. Právě tato metodika je zcela odlišná od tradiční. V XP se celý proces vývoje software točí pořád dokola, a opakuje se v jednom cyklu, která spočívá v několika fází, které se opakují. Totiž není to tak, že by fáze na sebe navazovali, nebo bylo zapotřebí na něco čekat. Každý den se celý team sejde, a několik časových jednotek stráví prací v jedné fázi, poté se přesune k druhé a k třetí, a takto dokola. Je zde hodně kladen důraz na teamovou komunikaci a spolupráci. Dále je zde důraz na komunikaci se zákazníkem. Hlavními znaky mohou být jednoduchý design a refactoring [3].
Výhody XP:

- Malé zatížení na menší projekty
- Zaručená kvalita díky mnoha testům po každé fázi
- Iterativní

Nevýhody

- Obtížné použít pro velké projekty
- Jsou zapotřebí zkušenosti a dovednosti při častém opravování kódu (rychlost, funkčnost) [2]

2.2.2 Scrum

Na rozdíl od ostatních metodik, Scrum v podstatě nenabízí žádné přesné vývojové metody, jako spíš nástroje, které se v daných fází mohou použít. [3] Dá se říci, že celý Scrum je založený na dobré dokumentaci po každém Sprintu. Sprint je většinou 30 denní procedura, při které probíhá vývoj systému. Na konci každého sprintu potom přichází dokumentace, a zjištění, co je dál potřeba udělat. Na začátku každé iterace je setkání s vedoucím teamu a zákazníkem, kde se řeší další postup. Zároveň se přibližně každých 15 minut team sejde k diskuzi, na jaký problém se narazilo a jak ho řešit [3].

2.2.3 Feature Driven Development

Na rozdíl od všech ostatních již jmenovaných metod, FDD se nezaměřuje na celý softwarový vývoj, ale spíše na fázi návrhu a vývoje. Celkově se FDD skládá z pěti fázích, přičemž prvními třemi fázemi jsou: Vývoj celkového modelu, Sestavení listu vlastností, plánování dle vlastností. Posledními dvěma fázemi jsou fáze Návrhu dle vlastností a Stavění dle vlastností. Poslední dvě fáze jsou právě definicí Agilní metodiky, jelikož jsou iterativní a rychle se v nich adaptuje k pozdním změnám v požadavcích [3].
V první fázi se provede celkový průchod systémem a jeho kontextem. Jsou zde vytvořeny funkcionální specifikace. Dále v listu vlastností se sestaví list, který bude

obsahovat vlastnosti pro podporu požadavků. V plánovaní podle vlastností se nastaví každé vlastnosti priorita. Následně jsou přesunuty jednotlivým vedoucím programátorům. Nakonec v posledních dvou fázích jsou brány vlastnosti dle priorit, kde jsou postupně naprogramovány. Po naprogramování všech vlastností, jsou všechny části seskládány dohromady [3].

2.3 Vyhodnocení Tradiční a Agilní metodiky

Agilní metodika je dle mého názoru lepším způsobem SDLC. Jelikož Tradiční metodiky mají dlouhou historii, takže lze předpokládat, že prošly větším vývojem, tedy bylo dáno více zpětných vazeb a tím větší prostor ke zlepšení. Na druhou stranu se agilní metodiky snaží být více flexibilní, což je jejich velkou výhodou. Co se týče počátečních požadavků od zákazníka, agilní metodiky umožňují iterativní komunikaci a postupné vylepšování, kdežto u tradičních metodik je zapotřebí vědět vše přesně dopředu, hned v první fází. Tedy ještě před vývojem a kódováním. Z toho vyplývá, že v agilních metodikách není obtížně přepracovávat různé fáze, naopak v tradičních je to velký problém a často zabere hodně času a práce. Z toho dále vyplývá, že u tradičních metodik máme fixní směr vývoje, kdežto opět u agilních metodik máme možnost jednoduché změny. Testování je poté opět lepší u agilních metodik, jelikož se provádí většinou na konci každé fáze (iterace). U tradičních se testování provádí až na konci kódovaní, tedy až na samotném výrobku. Jelikož agilní vývoj nabízí tolik možností co se týče vracení ve fázích, je zapotřebí, aby vývojáři byli zkušení a musejí mít větší znalosti. Právě proto, že se musí lehce adaptovat situaci, různým změnám v kódu. Z toho všeho vyplývá, že právě agilní metodiky se hodí spíše pro menší projekty, naopak tradiční pro větší.

2.4 Odchytávání chyb

Odchytávání a následné odstraňování chyb je na denním pořádku ve vývoji aplikací, ať už to jsou webové, mobilní nebo desktopové aplikace. Pro jednoduché školní programy si většinou vystačíme s odchytáváním výjimek. Výjimka (exception) je chybový stav (událost), který by neměl nastat. Pokud nastane, v podstatě narušuje běžný chod aplikace a program ji musí umět vyřešit. Pokud nastane chyba metody tak daná metoda vytvoří objekt, který se nazývá chybový objekt a je předán systému. Chybový objekt v sobě obsahuje informaci o chybě, která nastala. Tomu ději říkáme že metoda vyhazuje výjimku. V momentě, kdy metoda vyhazuje výjimku se systém snaží výjimku řešit. A zde právě přichází na řadu odpovědnost programátora, který to řeší. Správně by k tomuto stavu nemělo nikdy dojít. Říkáme tedy, že programátor odchytává výjimky, a určí, co se stane. Nikdy se nesmí stát, že po nějaké výjimce se aplikace ukončí. To je ta nejhorší
situace, která může nastat.

Pro odchytávání chyb existuje mnoho programů, ať už placených či zdarma, které se používají. Těmto programům se říká **Bug and issue tracking tools**. Tyto nástroje pomáhají vývojářům detekovat chyby, určit jejich přesnou pozici, zjistit kdy nastaly a za jaké události. Nástroje mohou události logovat, určovat priority chyby a podle toho se rozhodovat, jak bude s chybou dále zacházeno. Při některé velké chybě může přijít oznámení e-mailem, nebo jiným prostředkem. Používání těchto nástrojů se používá hlavně u větších aplikací, kde je potřeba velká kontrola nad mnoha segmenty aplikace, ale i v menších aplikacích. Hlavní výhodou je ulehčení hledání chyb, to znamená ušetření času, který je vždy velice cenný.

Jak jsem již zmínil, existuje mnoho placených i zdarma použitelných nástrojů. Pro zkušební účely jsem si vybral nástroj Log4net. Více o tomto nástroji bude zmíněno v kapitole 4.2.

2.5 Diagramy

V softwarovém inženýrství hrají diagramy velkou roli hlavně při návrhu a popisu aplikace nebo při celkovém chodu firmy a teamu. Z toho vyplývá, že diagramů je velké množství a každý z nich se dá používat na popis něčeho jiného. Prvním příkladem využití diagramů je popsání Business Process. Tyto procesy jsou aktivity nebo úkoly teamu, které vedou k docílení nějakého úkolu. Tedy vytvořením diagramu Business Processu získáváme snazší implementaci, tedy snazší pochopení problému a tím pádem získáme lehčí cestu k vyřešení problému. Řekl bych druhým nejdůležitějším typem jsou takzvané use case diagramy. Jedná se tedy o způsob identifikace a organizace požadavků vyvíjené aplikace. V zásadě se use case diagram vytváří jako první, protože díky němu dochází k lepšímu objasnění problému. Jedná se o jednoduchou grafickou implementaci, kde je dobře vidět, jak bude systém fungovat.

Dalším typem diagramů jsou vývojové diagramy. Vývojové diagramy určitě patří k vývoji jakékoliv aplikace, ať už webové nebo desktopové. Právě díky těmto diagramům můžeme přehledně navrhnout strukturu vývoje a chod aplikace. Vývojové diagramy se právě používají ve fázi návrhu. V softwarovém inženýrství se nejvíce používají takzvané UML diagramy. UML je unifikovaný modelovací jazyk, který právě pomáhá vizualizaci návrhu aplikace. Než se team pustí do kódovaní, nejlepší způsob jak jim předat velké množství informací o chodu aplikace, účelu a funkci je právě skrze UML diagramy. V UML diagramu figurují tvary objektů. Každý tvar má svou specifickou funkci. Například hranatý čtverec označuje třídu. Kosočtverec označuje větvení (podmínky if). Mezi objekty jsou vazby, které mohou být znázorněny více druhy čar (plná, čárkovaná, ...) a ty mohou být zakončeny šipkami, které označují směr chodu závislostí.

Vytváření správných a čitelných UML diagramů není jednoduché pro rozsáhlé aplikace, proto existuje spousta nástrojů, které nám usnadňují modelování. Nástroje však nemusí být pouze externí programy, velké množství vývojových prostředí dokáže

vygenerovat z kódu UML diagram. Generování UML diagramů z kódu je velice dobrá vlastnost vývojového prostředí, jelikož tato vlastnost ulehčí vývojáři práci a zdokumentuje celý software za nás. Ovšem existuje více způsobů a jedním z nich jsou právě diagramy. Ty nám totiž umožní vidět kód z grafického hlediska a pro dalšího programátora to je výstižný náhled toho, jak program funguje. A ne jen pro programátora, ale protože číst tyto diagramy není úplně složité, tedy slouží i pro zákazníka.

Entity Relationship Diagram dále jen ERD se používá v databázových modelech k popsání stavů mezi tabulky. V podstatě můžeme říci, že ERD nám znázorňuje logickou strukturu databáze.
Dalším typem je například Data Flow Diagram dále jen DFD. Ten se také používá již v návrhu systému, kdy se určují vstupy a výstupy aplikace. Jak název již naznačuje, jde o popsání toku dat, která budou aplikací procházet. Tedy například jestli bude aplikace načítat nějaký textový soubor dat a jestli bude ukládat nějaká data na webový server nebo někam disk a jak to bude dělat. Posledním příkladem je State Transition Diagram. Ty se opět používají hned od začátku při návrhu. Tyto diagramy nám popisují stavy objektů při reakci na nějakou událost. Velkou nevýhodou těchto diagramů je skutečnost, že pro správnou funkcionalitu je zapotřebí prokrýt všechny možné stavy systému. Tedy nesmí se stát, že objekt dostane hodnotu, která není popsána v diagramu, a dále nastane chyba, protože vývojář tuto hodnotu vůbec neočekával.

2.6 Logging

Logování se používá v širokém měřítku napříč mnoha aplikacemi. Je to způsob, kterým se desktopové aplikace sledují. Sledováním aplikací je myšleno sledování jejich neočekávaných chování, při kterých dochází k událostem. Tyto události jsou potřeba zalogovat, a podle důležitosti oznámit správce o této události. Ovšem příliš logování aplikaci škodí a naopak málo také. Některé logování zabírá příliš mnoho času. Pokud se objeví výjimka, která má hlubší závislost a má vysokou prioritu, je zapotřebí tuto výjimku například odeslat e-mailem vývojáři. Velice špatnou situací je stav, kdy těchto výjimek nastane za sebou více. Při slabším výkonu může dojít k několika vteřinovému pozastavení aplikace.
Logování lze provádět několika způsoby ukládání. Úplně nejjednodušším pro testovací účely je logování do příkazové řádky vývojového prostředí. To slouží vyloženě pro testovací účely nástroje. Další možností je logování chyb souboru, která je zároveň best practice. Soubor může být uložený v místě instalace aplikace a ukládáme do něj vše potřebné při události. Tím je myšlen datum, čas, důležitost, na kterém řádku nebo v jaké metodě a v neposlední řadě k jaké události došlo. Těchto způsobů logování existuje velké množství a dále jsou nejčastějšími možnostmi uložení do databáze nebo e-mailu. V závěru není potřeba ukládat logy do souboru, ale můžeme si pro ně vytvořit celou databázi a v případě vážné výjimky můžeme obeznámit developera e-mailem.
Následně se logování dělí podle důležitosti události (levelu). Z pravidla se dělí na tyto

levely: Debug, Info, Warning, Error, Fatal. Tyto levely jsou seřazeny dle důležitosti a samozřejmě se mohou v každém nástroji lehce lišit.
Logování umožňují velké množství nástrojů pro různé programovací jazyky, avšak je zde možnost, že sám programovací jazyk s některým frameworkem logování umožňuje taktéž. V mém případě .NET Framework v sobě má třídy, které logování umožňují. Více o těchto nástrojích a třídách je popsáno v kapitole 4.2.
Je velice důležité vědět, co logovat. Logovat je zapotřebí správným způsobem, pakliže budeme logovat přiliš mnoho, může dojít k chybám v systému. Příliš časté logování může zapříčinit systémové zahlcení (I/O operace) a může zabírat větší množství úložního prostoru. Dále příliš časté logování zapříčíní ztráty redundance dat nebo dokonce zbytečnost logů. To má vliv potom na jejich kontrolu a další ukládání. Pokud se musí procházet velké množství logů, zabere to více času, přičemž by mohlo dojít k situaci, že by některé logy mohly být zbytečné. V poslední řadě jde o přidávání bloků pro logování do kódu. To zvětšuje velikost kódu a je zapotřebí o logování vědět již při vývoji aplikace.
Z pravidla se místa nebo bloky kódu potřebná k logování určují při vývoji. Zároveň není přesně dáno, kolik by se mělo monitorovat částí aplikace a kolik naopak ne. Ve výsledku jde o to, jak si to vývojář určí, nebo je po něm žádáno od zákazníka.

3 Instalace a aktualizace aplikací

V tomto tématu se budu zabývat poslední fází vývoje životního cyklu aplikace a to je deployment. Tedy vytvoření instalačního souboru a distribuce ke koncovým uživatelům. Dále také způsoby, jak aplikaci aktualizovat a opravovat.

.NET Framework

.NET Framework je softwarové rozhraní primárně pro aplikace běžící na systému Microsoft Windows, protože právě Microsoft toto prostředí vytvořil. Jeho největší předností je skutečnost, že obsahuje knihovnu Framework Class Library, která nám umožňuje naprogramovat aplikaci ve více programovacích jazycích s tím, že zachová její
funkčnost. To znamená, že aplikace může být napsána například v Jave a v C++, ale bude stále vykonávat to samé nad podobnými datovými strukturami.
Microsoft začal vyvíjet .NET Framework v roce 1990. První verze byla .NET 1.0, která vyšla v roce 2000. Nejnovější verzí je 4.7.2, která vyšla 30.4. 2018 pro Visual Studio 2017. Verze 4.7.2 nahrazuje verze 4.0-4.7.1 [10].

Požadavky na deployment aplikací

V zásadě největším požadavkem je jednoduchý a spolehlivý instalační soubor. Jednoduchý znamená pro uživatele co nejvíce pohodlný, bez jeho většího zapojení. Zde se tedy řeší tiché instalace, kde uživatel vidí může vidět pouze progressbar (stav instalace) a bez jakého dalšího zásahu se aplikace nainstaluje. Oproti tomu je klasická instalace, kde uživatele „klikacím" způsobem aplikaci nainstaluje. Dále spolehlivý instalační soubor znamená stav, kdy instalační soubor i s jeho moduly je plně podepsaný. Pokud je spuštěn instalační soubor, který není podepsaný certifikátem, zobrazí se dialogové okno hlásící neznámého vydavatele aplikace. Pokud je instalační soubor podepsán, toto okno se neobjeví. Ovšem není podmínkou, pokud je instalační soubor podepsaný, že se jedná o bezpečnou aplikaci. Certifikáty mohou být podvrženy, či vytvořeny podvodné.

3.1 Podpisy kódu

Je to metoda, která používá certifikační digitální podpisy pro podpis spustitelných souborů (exe, msi), pro ověření identity autora (společnosti) dané aplikace a zajištění

toho, že kód nebyl změněn od té doby, co ji podepsal autor. Uživatelé se následně mohou lépe rozhodnout, zda mají aplikaci věřit, či ne.

3.1.1 Účel podpisů

Jedním slovem, důvěryhodnost. Pokud instalovaná aplikace, která je podepsaná firmou Microsoft nebo Apple, bude chápána jako bezpečná aplikace právě kvůli jejímu certifikátu. Pakliže aplikace nemá žádný podpis, nebo je něco podezřelého, je už jen na uživateli, jak dané aplikaci bude věřit.

3.1.2 Certifikáty na podpisy kódů

Tyto certifikáty nám umožňují podepisovat kódy pomocí soukromého a veřejného klíče. Tyto klíče jsou generovány, když jsou certifikáty vyžádány. Soukromý klíč je pouze uchován v počítači žádaného o certifikát. Veřejný klíč je posílaný zpět k poskytovateli klíče s certifikátem.

3.1.3 Důvěra podpisů

Podpisy by měly zprostředkovávat důvěryhodné a ověřené společnosti a autority, které používají infrastrukturu veřejných klíčů (PKI). Tyto společnosti ověřují celkovou správnost certifikátů. To ale neznamená, že samotnému kódu můžeme věřit. Téměř kdokoliv může certifikát vytvořit a podepsat škodlivý kód. [11]

3.1.4 Jak to funguje

Pro testové účely nám některé nástroje umožňují vytvořit testovací certifikát, který je „jen na oko“. V opačném případě musíme žádat společnost o certifikát, poté je jeho identita ověřena a obdrží certifikát, dále se vygeneruje jednostranný hash který se spolu s certifikátem přidají ke spustitelnému souboru. Jakmile uživatel obdrží aplikaci, ověřuje se identita například pomocí de-kryptovávání hashe pomocí veřejného klíče v certifikátu. [11]
Dále je zapotřebí řešit oprávnění správce při spouštění instalačního souboru. To může být zpravidla méně přívětivé pro uživatele, ale často to může být i zapotřebí.

Požadavky na monitorování aplikací

Nejprve je zapotřebí si uvědomit, o jaké aplikace se jedná. U webových aplikací je možné monitorovat jejich performace, konzistenci dat a celkovou komunikaci se serverem. V mé práci se zaměřuji na desktopové aplikace, kde se řeší monitoring jako sledování chyb a jejich logování. Zde jsou tedy požadavky v podobě vytváření logů na různých místech (cloud, e-mail, soubory). Sledování chyb a jejich logování je popsáno v kapitole 2.6. U webových aplikací se zároveň logování také řeší. Dalším způsobem je možnost sledování desktopových aplikací pomocí profilerů, které sledují jejich performance (zatížení procesoru, disku, RAM). Profilling umožňují již některé

vývojové prostředí, jako například Visual Studio. Profillingem jsem se v mé práci nezabýval, jelikož každý vývojař může otestovat běh aplikace při vývoji (na slabém a silném stroji).

Požadavky na ladění a údržbu

Jak bude dále zmíněno v kapitole 4.1, ladění a údržba se zastřešuje pod aktualizaci aplikací a logování. Pomocí logování jsou zjišťovány potřebné a důležité informace o aplikaci, dále o uživateli, jak s aplikací zachází a podle těchto záležitostí je rozhodováno, jak dále s aplikací bude naloženo. Tím je myšleno jak a kdy budeme vydávat aktualizace, jak budou komunikovat s uživatelem a hlavně co se v daných aktualizací bude řešit. Pokud je zjištěno, že například uživatel často nekliká na tlačítko které by měl, je možné ho přesunout nebo více zvýraznit. Sledovat klikání uživatele můžou zajistit například countery, které čítají klikání na jednotlivá tlačítka, nebo měřit čas strávený v jednotlivých částech aplikací, a poté tyto informace například logovat každý týden s jejich celkovým počtem. Pokud je zjištěno, že aplikace hlásí chybu v nějaké části, nebo se nějaký segment nezobrazuje jak má, opět vydáme aktualizaci, která tento problém řeší.

Ladění a údržba spojená s profilingem

Při zkoumání problematiky deploymentu bylo zjištěno, že monitoring u desktopových aplikací se řeší logováním 2.6 a dále pokud nás zajímá performance, tak jsou použity profilingový nástroje 3.1.4.

3.2 Instalace aplikací

Při řešení instalací aplikací dochází k mnoha problémům, zejména problémům u koncového uživatele, které jsou potřeba vyřešit. Opět se vracíme k fázi návrhu, je potřeba znát, k jakým koncovým uživatelům bude aplikace doručena a od toho se následně odvíjí instalační program. Jelikož instalační soubor může být více druhů (exe, msi a další).

3.2.1 Problematika instalace aplikací

První rozhodnutí nastává ve fázi designu, kdy je potřeba vědět, na které koncové zařízení bude aplikace patřit. Řekněme pokud aplikace bude sloužit pouze pro systém Windows, je poté jednoduché vybrat typ instalačního souboru (například msi). Pokud bude možné aplikaci instalovat na více platformách, je potřeba udělat více druhů instalačních programů dle platforem. Msi je instalační formát od Microsoft Windows, který je v dnešní době doporučeným formátem pro vytváření instalačních souborů pro systém Windows. V zásadě jsou to databázové soubory, které využívá Windows Installer. Dá se říci, že msi soubory jsou spouštěny exe souborem, který se ve Windows nazývá MSIEXEC.exe.

Při instalaci je dobré zkontrolovat, zda aplikační požadavky na systém jsou splněny. Například některé hry mohou vyžadovat silné grafické karty nebo procesor. Zkontrolování zda již software v počítači existuje, popřípadě jaká verze. Dále se při instalaci aplikace často neinstaluje pouze jedna část, ale přidává se do systému konfigurační soubory aplikace, registry a někdy i samotné proměnné (environment variables), které mohou ovlivnit jak bude proces fungovat na systému. Přidávání možností pro více uživatelů v počítači, pro skupiny a podobně.

3.2.2 Způsoby instalace

Instalovat aplikace lze mnoha způsoby a v některých případech není vůbec potřeba zásah uživatele k instalaci. Prvním způsobem je klasická instalace s průvodcem. Tedy uživatel stáhne instalační soubor z internetu (nebo z datového média) a pomocí instalačního průvodce se „dokliká" až na konec instalace. Tento způsob je samozřejmě nejobecnějším způsobem, který je dost používaný v dnešní době. Druhým způsobem je **tichá instalace**. Tichá instalace je způsob instalace, kdy není po uživateli žádána žádná interakce kromě spuštění samotného instalátoru. Jedním z důvodů, proč používat právě tento způsob je celková automatizace při instalaci na více systémech. Tyto instalace mohou být dávkově spuštěny přes příkazový řádek na více systémech, a uživatel u toho nemusí být.
Dalším způsobem je instalace po síti. Tento způsob se používá ve společnostech v kombinaci například s tichou instalací. Na lokálním serveru je uložen instalační soubor, který je spuštěn a rozeslán na více stanic zároveň. Ve společnostech se používá kombinace tiché instalace například přes lokální síť a zároveň s takzvanou plánovanou instalací. Jak již název napovídá, instalace je naplánována časově dopředu. Systémový administrátor naplánuje pomocí scheduleru instalaci na nějaký čas, kdy se na stanicích nepracuje. Zároveň je potřeba brát ohled na bezpečnost. Pokud se instalace z nějakého důvodu nezdaří, je potřeba vrátit se o verzi zpět, nebo opakovat pokus instalace. Tiché instalace jsou v celku dobrou metodou pro automatizaci procesu, ale pro běžné uživatele je výhodnější použít standardní instalační soubory, kde si sami vyberou místo, kam se aplikace nainstaluje, jazyk aplikace, pokud jej aplikace podporuje a celkově vidí samotný průběh instalace. Je to ve výsledku lehce delší proces, ale lze sledovat celkový průběh instalace. Pokud při tiché instalaci nastane chyba, uživatel si toho nemusí všimnout.

Prerekvizity

Prerekvizity jsou častou podmínkou úspěšné instalace. Příkladem může být nainstalovaný .NET Framework. V případě, že prerekvizita není nainstalovaná, instalační program by měl daný framework nainstalovat. V případě, že podmínky nejsou splněny a nelze framework doinstalovat, instalace by měla selhat. Nemusí se jednat pouze o framework, ale o spoustu jiných nainstalovaných programů, verze systému, potřebný hardware a podobně. Použití prerekvizit v této práci bude popsáno v kapitole 3.6.

Automatizace vytváření instalačních souborů

Celý svět spěje k celkové automatizaci, takže ani instalace aplikací nezaostává. K docílení vytvoření instalačního souboru lze dvěma způsoby. Prvním a zároveň tím nejvíce běžným a nejvíce používaným je vytváření přes GUI. V dnešní době je na trhu mnoho nástrojů, které instalační soubory vytvářejí, ale jen málo z nich umožňuje celý proces zautomatizovat. A na to se právě v mé práci zaměřuji.
Celý proces automatizace spočívá ve vytvoření instalačního souboru v jeden daný čas, například ve windows scheduleru. Prvním krokem k cílení tohoto úkolu je najít potřebný nástroj, který umožňuje vytváření instalačních souborů ze skriptu. To znamená vytvoření bat skriptu, který bude následně daný čas spuštěn v příkazové řádce, a provede kompletní vytvoření instalace. Zároveň musí nástroj umožňovat nastavovat a konfigurovat následně vytvořený instalační soubor v programovacím nebo značkovacím jazyce. Nejčastěji se právě jedná o XML. Jelikož celkový proces automatizace je složitý, je dobré, aby daný nástroj zároveň umožňoval podepisování aplikací svým vlastním nástrojem. Došlo by poté k zjednodušení celého procesu.

3.3 Aktualizace aplikací

Aktualizace aplikací je proces, kdy je zapotřebí vydat novou verzi aplikace. Důvodů může být více, například opravy chyb, vylepšený UI nebo bezpečnost. Aplikace se aktualizují ve většině příkladů automaticky. V této době se aktualizace provádějí přes internet, kdy při spuštění aplikace nastane situace, že launcher aplikace zjistí novou dostupnou verzi na serveru a pokusí se jí stáhnout. V některých případech není zapotřebí zásah uživatele. Tedy tiché aktualizace. Ty se provádí v případech velmi malých oprav, které neomezí provoz aplikace. V případě velkých aktualizací je zapotřebí aplikaci vypnout, nainstalovat novou verzi a opět zapnout. S tímto přichází ale velká spousta otázek, například kam stahovat a instalovat nové verze, vytváření záloh pro obnovu, kontrola správnosti nainstalované verze a podobně. O těchto problémech a způsobech se budu nadále zabývat ve vybraných dvou způsobech instalací a aktualizací aplikací.

Jak na automatické aktualizace

Prvním případem je stav, kdy je potřeba aktualizovat aplikaci v okamžiku, kdy je vydaná a dostupná ke stažení nová verze. K docílení tohoto případu je zapotřebí aplikaci, která hledá novou verzi průběžně. Takovou aplikaci je možné naprogramovat, což se spíše nedělá jelikož to není jednoduché, a nebo to už umí sám daný nástroj, který je použit pro vytvoření instalačního balíčku. Dále potřebuje místo, kam se budou aktualizace stahovat.

Systém pro aktualizaci

Pro automatickou aktualizaci se nesmí použít takový systém, které používá různé služby, protože služby vyžadují oprávnění správce a to by ve výsledku ztrácelo celý

smysl. Musí se brát ohled, kam se bude aktualizace stahovat, jestli bude proces aktualizace potřebovat některá oprávnění a jak daný systém vyhodnotí, zda je potřeba aktualizaci nainstalovat nebo ne (nevhodný čas, přehlcená síť, méně důležitá aktualizace, aktualizace pro jiné cílové počítače, ...).

Samotné aktualizace

Co je potřeba zmínit na úvod je situace, kdy aktualizace by neměly být tam, kde jsou soubory aplikace (exe). Aktualizace fungují ve dvou směrech. Významná a méně významná aktualizace. Významná aktualizace (Major upgrade) funguje takovým způsobem, že se nejdříve stará verze odinstaluje a nainstaluje se nová. Méně významná aktualizace (Minor upgrade) funguje tak, že se pouze stará verze aktualizuje (patch).

Nyní je však potřebné se rozhodnout, jak se aktualizace budou stahovat a jak se zjistí, že je potřeba stáhnout nová aktualizace. Právě toto řeší programy třetí strany (ClickOnce).

Automatické aktualizace umí nástroje msbuild, WiX, AdvancedInstaller, Click Once, které budu popisovat v kapitle 3.6 a 3.7 zároveň s klasickými „klikacími“ a stahovacími způsoby.

Použití při testování

Při testování byl použit systém přímo integrovaný v samotných nástrojích, které byly použity pro vytváření instalačních souborů. V kapitole 3.6 a 3.7 je dále popisováno více informací o nich. Oba zkoumané nástroje umožňují aplikaci aktualizovat, jen msbuild situaci zvládá lépe, než WiX Toolset.

3.4 Microsoft Visual Studio 2017 Installer Projects

Velice jednoduchý a přehledný doplněk k Visual Studiu. Tento doplněk je plně zdarma bez žádných omezení. Zároveň neřeší automatické aktualizace. Pro vytvoření instalátoru stačí vytvořit window forms application a poté vytvořit nový projekt - jiné projektové typy - Visual Studio Installer.

3.4.1 Výsledek po testování

Při instalaci je nutné ověření správce. Instalátor lze mít v jiném jazyku (i český jazyk). Zároveň lze instalaci provést do různých složek bez problému. Aktualizace opět bez problému, proběhla úspěšně, i když jsem nezměnil kód. Nevýhoda je, že nelze přidávat hodnoty do registrů, ani nepodporuje tichou instalaci. Lze mít instalaci ve více jazycích, i vlastní prostředí. Lze přidat kontrola .NETu, zda je nainstalovaný.
Ve výsledku největší výhodou nástroje je jeho dostupnost, jelikož je zdarma a je doplněk do Visual Studia. Bylo zjištěno, že je toto rozšíření spíše pro zkušební účely, pro testování, protože neumožňuje takové služby jako nástroje popisované dále a celkový proces je velice zdlouhavý. (viz. mazání starších verzí).

Závěrem bylo zjištěno, že se doplněk hodí pro testování a zkušební účely, protože instalaci a aktualizace umožňuje bez problému, ale jen omezeně.

3.5 Advanced Installer

Placená i free verze (30 denní trial). Je zde možnost samostatné aplikace nebo jako doplněk do Visual Studia. Aplikace umožňuje upravit instalátor, různé loga, obrázky, celkový vzhled, barvy, písmo. Je zde dost možností z čeho si vybírat a i přesto je poté možnost si daný instalátor upravit dle svého mínění, tím je myšleno svoje vlastní obrázky, loga a další. Zároveň umožňuje automatické aktualizace a tiché instalace. Je to velice rozšířený program, který podporuje velkou škálu verzí .NETu, různé balíčky, podporuje i více programovacích jazyků a práci s nimi. Také umožňuje SQL browsing, práci s XML soubory, konvertování z msi do App-V a další. Instalátor nabízí možnost analyzování a je zde možnost patchování.
Přestože má tento program velkou škálu možných úprav a vylepšení, skoro všechny jsou možné jen v placené verzi. Ve volné verzi je možné si vybrat pouze jeden vzhled, nelze nastavit auto updater, licence a lze vytvořit pouze msi instalátor. Naopak placená verze umožňuje msi, exe, CD/DVD, silent MSI, Web installer.

3.5.1 Výsledek po testování

Vše ve volné verzi funguje bez problému. Instalace do různých adresářů, nezáleželo na verzi aplikace. Nová verze aplikace lze kontrolovat z webových stránek. Je potřeba oprávnění správce, to lze ovšem vypnout, pokud se instalace týká pouze aktuálního přihlášeného uživatele. Při tiché instalaci se aplikace nainstaluje do AppData (lze opět bez oprávnění správce). Po spuštění se objeví malé okno, po skončení zmizí a ukončení proběhne bez oznámení. Umožňuje automatické aktualizace, kontrolu frameworku, podpora více jazyků a vlastní UI.
Na závěr bych řekl, že tento program na mě působil velice pozitivně. Ve výsledku byl tento nástroj vyhodnocen velice kladně. Je velice přehledný, jednoduchý a efektivní. Umožňuje vše co je potřeba, akorát jen v placené verzi. Tedy je doporučen jak pro pokusy, tak i na pokročilé úrovni.

3.6 WiX Toolset

Prvním vybraným nástrojem pro vytváření instalačních balíčku s možností automatizace je WiX Toolset. WiX je nástroj, který se používá na platformě Windows a jádrem tohoto nástroje je jazyk XML. Tento nástroj umožňuje vytvářet instalační balíčky pomocí jazyka XML, který je velice jednoduchý a přehledný pro použití. WiX Toolset dobře funguje s Visual Studiem, buď jak samotný rozšiřující nástroj, nebo jako samostatný program. Je velice používaný napříč světem a je zároveň zcela zdarma. Na oficiálních stránkách lze nalézt kompletní dokumentaci k nástroji a dále odkaz na portál github, kde probíhají diskuze o chybách a nových verzích. Jedním

z hlavních důvodů, proč jsem si vybral tento nástroj je automatizace instalačních souborů.

Vlastnosti

WiX Toolset nabízí možnost změny UI instalačního programu, stejně jako změnu jazyka. Jazyk se dá nastavit podle jazyka uživatelského systému, na který je program instalován. Změna UI může být provedena vybráním předem nastavených vzhledů (celkem 5). Je zde ovšem možnost vytvoření vlastních vzhledů, což ale nese nevýhodu. Všechny obrázky musí být formátu bitmap. Jednou z největší vlastností je skriptování. Skriptování je způsob vytváření instalačních souborů pomocí skriptů, které se dají spouštět dávkově pomocí příkazové řádky. To přináší zjednodušení neboli automatizaci, jelikož administrátor sítě může nastavit čas v scheduleru, který spustí skript. V tomto skriptu je možné nastavit dodatečné informace ohledně instalačního souboru. Zároveň nabízí možnost tichých instalací. Tím pádem není potřeba oprávnění správce při instalaci. Více podrobnějších informací lze naleznout [9].

Výsledek po testování

První na řadu přichází XML soubor, který byl zapotřebí vytvořit a přidat k projektu. XML soubor obsahuje základní kostru instalace, která je určená pro základní funkcionalitu. Vše je dobře popsáno v dokumentaci na oficiálních stránkách, jak pustupně celý XML projít a co kde vytvořit. Přidáváním XML tagů jsem se postupně dostal k vytvoření instalace s jiným jazykem a jiným vzhledem. Část psaní XML souboru je jednoduchá a poté vytvoření instalace také, pokud se jedná o jednoduchý instalační soubor bez žádných doplňků. To se provádí pomocí sestavení WiX souboru (Build). Poté se nám instalační soubor vytvoří do složky, ve které máme uložený projekt, ke kterému vytváříme instalační soubor. Pro vylepšené funkce instalátoru je potřeba přidat pomocí Referencí jednotlivé knihovny WiXi k WiX souboru, který upravujeme. Tím je namysli funkce vlastního UI, možnost výběru kam se nainstaluje aplikace a podobně. Vše opět jednoduché a dobře zdokumentované. První problém nastal při podepisování aplikace. Podepisování a celkově certifikáty jsou velice důležité v dnešní době pro jakékoliv aplikace. Slouží pro přidání důvěryhodnosti aplikace. Certifikáty zaručují, že aplikaci vydala důvěryhodný zdroj. Ovšem to neznamená, že pokud bude aplikace podepsaná, jedná se o aplikaci bezpečnou k nainstalování. Certifikát může být vytvořen jako plagiát. Pro účely v této práci byl vytvořen testovací certifikát pomocí nástroje ve Visual Studiu s názvem Click Once. O tomto nástroji bude více zmíněno v další kapitole, ale jeho největší výhodou je funkce, která umožňuje vytvořit certifikát pro testovací účely. Pro certifikáty WIX toolset používá nástroj od Microsoftu, který se nazývá signtool.exe. Tento nástroj se nachází v balíčku **Windows Kits**. WIX Toolset umožňuje dva způsoby, kterými se dá podepsat aplikace.

Prvním způsobem je přímo úprava XML souboru, který je dlé mého názoru zároveň nejsložitější. V XML souboru zapotřebí vytvořit cesty k souborům, které

jsou potřeba podepsat, ale i k podepisovaným nástrojům. V dokumentaci není tento postup popsán. Bohužel se mi tímto způsobem nepodařilo aplikaci podepsat. Cesty k souborům a nástrojům jsem měl nastaveny správně.
Druhým způsobem je využití nástroje **insignia**. Nyní se dostáváme k používání nástroje WiX Toolsetu. Nejprve popíši funkci nástrojů, které ve výsledku vedou k vytváření instalací pomocí skriptů a následně se dostanu k daným nástrojům.

Prerekvizity

Prerekvizitou bylo zkoušeno, zda je na počítači nainstalovaná potřebná verze .NET Frameworku. Jelikož tato práce pojednává o .NET aplikacích, je důležité aby při spuštění vytvořené aplikace bylo na počítači zkontrolováno, zda daný framework chybí, či naopak nechybí. Pokud nechybí, je dále potřeba zkontrolovat verze. Pokud chybí, je zapotřebí nainstalovat celý framework s potřebnou verzí ještě před nainstalováním aplikace. Pokud je zapotřebí pouze uživateli oznámit o chybějícím frameworku, stačí mu pouze vypsat hlášku, která oznámí že framework není nainstalován, a dále instalace selže. To lze docílit například tímto segmentem kódu

```
<PropertyRef Id="WIX_IS_NETFRAMEWORK_45_OR_LATER_INSTALLED"/>
<Condition Message="This application requires .NET Framework 4.5>
<![CDATA[Installed OR WIX_IS_NETFRAMEWORK_45_OR_LATER_INSTALLED]]>
</Condition>
```

Dále bylo zjištěno, že pokud je daný framework potřeba i nainstalovat, je nutné vytvořit bootstrapper. Bootstrapper je aplikace hlavního instalační souboru, který řeší vedlejší instalace. Na vytvoření bootstrapperu má WiX Toolset opět nástroj, který se nazývá burn. [8]. Ovšem to byl případ pro vytvoření prerekvizit, které umí sám WiX. Pokud se jedná o prerekvizity aplikací třetích stran, je postup lehce odlišný.

Pokud se jedná o program třetích stran, který je potřeba být prerekvizitou v instalačním souboru, lze opět použít nástroj burn. Jediný rozdíl je v konfiguraci XML dokumentu. Do té se musí přidat element PackageGroup a do něj ExePackage, ve kterém se definují a specifikují vlastnosti prerekvizity a zároveň její odkaz na stažení popřípadě už stažená aplikace, která se nainstaluje v případě, že na koncové stanici chybí.

Vytváření instalací pomocí skriptů

Pro vytváření instalačních souborů z příkazové řádky jsou nejdůležitější dva nástroje, tím jsou **Light a Candle**. Nejdříve se na WiX soubor, který patří k projektu, zavolá nástroj Light. Ten převede a připraví soubor k vytvoření instalačního souboru. Dále se zavolá nástroj Candle, který může obsahovat v příkazové řádce více argumentů, který již instalační soubor vytvoří tak jak je nadefinováno ve WiX souboru nebo popřípadě s přidanými parametry, které ho modifikují. Na další ukázce předvedu způsob použití.

```
%wix_dir%\light.exe  %src_dir%\Product.wixobj
%wix_dir%\candle.exe %src_dir%\Product.wxs
%wix_dir%\insignia.exe -im C:\Users\Lukáš\Product.msi
```

Text, který je uzavřen ve znacích % jsou lokálně vytvořené proměnné, které v sobě uchovávají cestu k nástrojům a k WiX souborům.
Podepisování souborů je složitá disciplína, ale zároveň lze docílit mnoha způsoby. Prvním způsobem je úprava XML souboru, která není úplně nejjednodušší. Dalším způsobem je využití výše zmíněného nástroje insignia, který funguje pouze na instalační souboru, které mají soubory pro instalace vložené v takzvaném cabinet adresáři. Tedy soubory, které se mají nainstalovat, jsou externě uložené mimo instalační soubor. Bohužel se mi tímto způsobem také nepodařilo dosáhnout cíle. Zvolil jsem tedy cestu, která využívá nástroj od společnosti Microsoft. Tento nástroj se nazývá SignTool a je součástí balíčku Microsoft Kits. Ten je potřeba si stáhnout, jelikož nepochází ze standardní instalace Windows. Abych ale mohl tento nástroj použít, je zapotřebí soubory pro instalaci zahrnout do instalačního souboru. Tedy aby nebyl potřeba cabinet. To lze docílit přidáním jednoho řádku kódu do XML souboru

```
<MediaTemplate EmbedCab="yes" />
```

Předtím, než je instalační soubor vytvořen, je zapotřebí všechny programy a knihovny podepsat dopředu. Totiž tento nástroj neumožňuje podepsat soubory tak, že podepíšeme pouze instalační soubor. Nejdřív tedy podepíšeme všechny komponenty zvlášť, poté vytvoříme instalační soubor, který podepíšeme stejným způsobem a nakonec aplikaci nainstalujeme. Níže lze vidět celý postup.

```
%sign_path%\signtool.exe sign /f %cert_path%\key.pfx
/p password %app_path%program.exe
%sign_path%\signtool.exe sign /f %cert_path%\key.pfx
/p password %app_path%library.dll
%sign_path%\signtool.exe sign /f %cert_path%\key.pfx
/p password %app_path%install.msi
```

Opět mezi % se nachází cesta k danému nástroji či souboru.

3.7 Nástroj msbuild

Tento nástroj je přímo integrován ve Visual Studiu. Není tedy potřeba žádné dodatečné stahování nebo instalace nástrojů či Windows Kit. Pro své testování jsem spojil msbuild s nástrojem Click Once, který je také součástí Visual Studia. Click Once je nástroj, kde se definuje funkcionalita instalačního souboru. Lze zde velice jednoduše řešit verzování, aktualizace ale hlavně lze zde velice jednoduše nahrát certifikát nebo dokonce vytvořit certifikát pro testovací účely. Vše se nachází pod záložkou Publish. Opět zde lze vytvořit „klikací“ instalační soubor, ale v mé práci se zaměřuji na automatizaci tohoto procesu, kterou samozřejmě msbuild nabízí. Postup je velice jednoduchý oproti WiXu. S instalací Visual Studia přichází nástroj

Visual Studio Command Prompt (nebo MSBuild Command Prompt). Po spuštění toho nástroje, jak už z jeho názvu vyplývá, se spustí příkazový řádek. V něm se stačí postupně přepnout až do adresáře, ve kterém se nachází náš projekt. Poté je zapotřebí na vytvořený projekt spustit nástroj msbuild.

```
msbuild /target:publish
```

Opět díky tiché instalaci není potřeba oprávnění správce, proces je jednoduchý, velké možnosti a jednoduché podepisování kódu. Největší nevýhodou je neschopnost vybrat si místo, kam se po instalaci aplikace uloží. Totiž msbuild aplikaci nainstaluje do AppData, kde právě není zapotřebí oprávnění správce. Tím se msbuild v podstatě chrání.

3.8 Vyhodnocení nástrojů pro deployment

Nejlepším nástrojem je při mém testování a pozorování nástroj WiX. Díky jeho bohatým možnostem vytvoření instalačních souborů a dále jeho konfigurování a spravování je vytváření instalací velice snadné, a existuje mnoho možností, jak takový soubor navrhnout. Zároveň bych ale oproti ostatním nástrojům neopovrhoval nástroj msbuild s použití ClickOnce, jelikož tím docílíme podobného výsledku. Ovšem ne tak dokonalého, ale pro testovací účely nebo pro menší projekty je to velice dostačující. Zároveň nabízí možnost vytvoření testovacích certifikátů, což se skvěle hodí k testování u msbuildu nebo tyto certifikáty můžeme použít i v jiných nástrojích. Dále bych také rád zmínil vcelku pěkné nastavení kontroly aktualizací, hlídání verze aplikace a celkově práci s aktualizacemi. Oproti tomu WiX Toolset tyto záležitosti automaticky nedělá, samozřejmě je potřeba instalační soubor nakonfigurovat tak, aby nové verze hlídal sám.
V práci byly zkoumány tiché instalace. Oba nástroje umožňují tichou instalaci vytvořit. V tomhle případě byl lepší nástroj msbuild, jelikož ten se na tiché instalace přímo zaměřuje. Naopak WiX Toolset umožňuje lépe vytvořit instalační soubor s GUI. Definování prerekvizit aplikací třetích stran a obecných prerekvizit umožňuje pouze WiX Toolset.

4 Monitoring aplikací

Monitoring aplikací v znamená právě sledování těchto aplikací nebo-li jejich kontrolování. V praxi se používá termín application monitoring a tím se zabývá Application Performance Managment. Jenže APM se zaměřuje převážně na webové aplikace a nebo aplikace, které komunikují se servery či databázemi. To má ovšem velice jednoduché odůvodnění. U těchto aplikací, které nějakým způsobem komunikují přes internet, lze monitorovat více věcí. Jelikož je potřeba sledovat internetový povoz, lze tedy měřit, jak rychle aplikace posílá data, jejich konzistentnost a spolehlivost. Dále jak aplikace funguje u klienta, její run time a podobně. U desktopových aplikací, které nepracují s internetem, není tolik co monitorovat kromě chyb aplikace. Samozřejmě je možné určitým způsobem sledovat performance desktopové aplikace, jenže to nemá velký smysl, protože vývojář při testování vyzkouší, jak aplikace běží na silném či slabém počítači nebo dokonce na počítačích, kde bude aplikace běžet, pokud jsou cílové počítače známi. Tedy ve své práci jsem se zaměřil právě na sledování chyb, které by mohly nastat v naší aplikaci.
Samotné sledování chyb už za nás ve větší míře dělá programovací jazyk. Ten totiž vyhazuje výjimky, které mohou nastat. Poté je potřeba tyto chyby logovat, aby vývojář věděl, že nějaká chyby nastala, s jak velkou prioritou a v jakém místě. Zde zmíněné logování za nás dělají nástroje třetích stran, které budou zmíněné v kapitole 4.2 nebo vlastní programovací jazyk.
Hlavním požadavkem na monitoring aplikací je sledování stavu výjimek v rizikových místech. My jsme totiž schopni odhadnout, kde by výjimka mohla nastat. V tom případě si ulehčíme práci, a můžeme se zaměřit pouze na konkrétní blok kódu. Další možností je sledovat úplně celou aplikaci, což může vést k tomu, že budeme muset logovat úplně vše, co nastane. A jak jsem se již zmínil v kapitole 2.6 není dobré logovat a sledovat vše co se děje. Nejlepší možností je logování těch nejkritičtějších chyb tak, že logger oznámí e-mailem vývojáři a další část se bude posílat na cloudové řešení a zbytek do souboru. Odesílání e-mailu může omezit chod aplikace, protože to zabere run time. Ukládání chyb do souboru při rozsáhlé aplikaci může vést k zabírání většího množství místa na disku. Ovšem teď se vracím k tomu, proč jsem popisoval metodiky vývoje aplikací. Dobře navržená a naprogramovaná aplikace má menší možnost pádu a výskytu chyby.

4.1 Ladění a údržba

Ladění a údržba je silně spojena s logováním. Zároveň dobře navržená a naprogramovaná aplikace má menší nároky na ladění a údržbu. Ladění a údržba může probíhat v měsíčních cyklech. Jedná se o takzvané patchování aplikace, což v podstatě znamená opravu chyb a poté se vydá záplata, která opravuje sadu chyb (aktualizace). Aktualizace aplikace se rozděluje na menší a větší aktualizace. Menší aktualizace může být prováděna ideálně v měsíčních cyklech a pokud je zapotřebí rychle opravit chybu, tak co nejdříve po záplatování. Menší aktualizace obsahují opravu kritických chyb pro chod aplikace, menší vylepšení jako například UI a podobně. Ze zásady tyto aplikace mají malou velikost. Pokud se vydávají menší aplikace, mění se pouze poslední čísla u verze aplikace.
Větší aktualizace přicházejí například jednou za rok, nebo za půl roku. Jsou rozsáhlé aktualizace, které mění například celý vzhled aplikace, přidávají velké množství nových funkcí a nástrojů pro uživatele a nebo nástroje pro vývojáře. S příchodem velké aktualizace se mění celá verze aplikace, to znamená že pokud máme verzi 1.0.0 s velkou aktualizací se z pravidla přepne na verzi 2.0.0.
Systém aktualizací se může lišit v různých prostředích a různých podmínkách. Samozřejmě se v dnešní době nejčastěji aktualizuje pomocí internetu. Postup je následující. Při vytváření instalačních souboru se k instalovaná aplikaci přidá kontroler, který hlídá verzi aplikace a verzi, kterou je možné stáhnout. Jednoduše hlídá pouze jejich čísla (verze). Pakliže se čísla lišej, je buď klient dotázán, jestli si přeje aktualizace nainstalovat, nebo je aktualizace nainstalována automaticky. Instalace aktualizací může být různá podle její velikosti. Pokud se jedná o menší aktualizaci, tak ta může probíhat za chodu aplikace, ale změny se projeví až po jejím restartu. Naopak při větších aktualizací je nejdříve potřeba aktualizaci stáhnout a nainstalovat, poté až aplikaci spustit. Samozřejmě pokud aktualizace selže, je zapotřebí se vrátit k předchozí stabilní verzi. Dobře navržený systém aktualizací by měl postupně předchozí verze a soubory mazat, pokud nejsou potřeba. Dochází totiž potom k zabírání místa na disku, což není úplně vhodné. Při instalaci se totiž vytváří dočasné soubory a soubory pro její kontrolu.
Aktualizace může hlídat samotná aplikace, nebo se k instalované aplikaci přibalí takzvaný hlídač, který běží na pozadí, pokud běží hlavní aplikace a hlídá její verzi. Aktualizace je možné kontrolovat pouze při spuštění nebo za běhu aplikace.
Jak jsem tedy zmínil, aktualizace probíhají kontrolou po síti, například z ftp serveru, nebo hlavního serveru poskytovatelů aplikace. K tomu je samozřejmě zapotřebí připojení k síti.

4.2 Vybrané nástroje pro monitoring, ladění a údržbu

V této kapitole popíši vybrané nástroje, kterými se budu zabývat a budu je zkoušet na ukázkových aplikací. Zároveň popíši i alternativní možnosti a také na jakých ukázkových aplikací je budu zkoušet a které vlastnosti na nich budu testovat. Všechny vybrané nástroje jsem vybíral dle hodnocení ostatním vývojářů a dle počtu stažení

například ve Visual Studiu.

4.2.1 Ukázkové aplikace a zkoušené vlastnosti nástrojů

Pro testovací účely jsem vytvořil aplikaci, na které zjišťuji vlastnosti nástrojů pro vytváření instalačních souborů a logování.
Aplikace obsahuje tlačítko, které po stisknu zaloguje testovací chyby do souboru a do cloudového řešení. Dále k instalačnímu souboru je zapotřebí připojit knihovny pro funkčnost logování. Zkoušené vlastnosti instalačních nástrojů:

- Tiché instalace vs instalace s GUI
- Oprávnění správce při spouštění instalace
- Podepisování instalačních souborů
- Podepisování jednotlivých modulů

Zkoušené vlastnosti logovacích nástrojů:

- Způsoby logování (konzole, soubor, cloud, ...)
- Složitost logování
- Údržba v kódu
- Omezení runtimu
- Kolik knihoven je potřeba připojit ke spustitelnému souboru

4.2.2 Log4net

Log4net je nástroj od Apache, který slouží pro logování. Jak z názvů napovídá, je určen pro .NET aplikace. Zároveň lze usoudit, že existuje pro více a platforem, například pro logovaní java aplikací se loger nazývá log4j. Dále existují logery pro PHP a C++. Apache disponuje zároveň s log viewerem, který se nazývá chainsaw a slouží jako GUI pro log4j.
Log4net je velice dobře zdokumentovaný, zdarma a velice rozšířený, takže jakákoliv chyba je pravděpodobně už monitorována nebo dokonce je vyřešená například na stackoverflow. V mé práci jsem log4net používal ve Visual Studiu jako doplněk k aplikaci, který lze nainstalovat přímo ve Visual Studiu přes nuget balíčky. Pouze ze statistik stahování tohoto nástroje je vidět, že patří mezi nejvíce používané nástroje. Počet instalací přes nuget balíčky činí totiž 15,4 milionu. Samozřejmě balíčky lze instalovat i přes Visual Studio cmd, takže toto číslo je pouze ilustrativní a může být větší. Specifické informace jsou dostupné v [7].
Po instalaci přes nuget balíčky se do projektu přidá konfigurační soubor s názvem log4net.config, ve kterém probíhá všechna konfigurace potřebná k logování. Další možností je konfigurovat log4net přímo v souboru App.config. V souboru se pracuje pomocí XML. Struktura je velice jednoduchá, a není potřeba znát příliš složité

konstrukce. Celý kód je obalen ve značce log4net a v něm se nachází pouze root, kde je řečeno, jaké appendery budeme používat, a dále samotné appendery, kde je nastaveno jejich použití. Appendery jsou elementy, které říkají jaký způsob logování se bude používat. Samozřejmě v jedné aplikace je možné použít více druhů appenderů. Log4net nabízí appendery jako výpis na standardní výstup Visual Studia, do souboru, databáze a e-mailem. To jsou standardní appendery od Apache. V každém appenderu je potřeba nastavit jméno, jak bude vypadat výstup po logování (formát), a dále potom vlastnosti, které si žádá každý appender. V případě logování do složky to může být název souboru a umístění. V případě e-mailu odesílatel, příjemce, nastavení protokolů, hesel a přihlašovacích jmen. Dále portů, předmětů a podobně. Zde ovšem nastává největší problém, kterým je právě posílání e-mailů.
Ve svých aplikací jsem zkoumal posílání e-mailové zprávy ze svého účtu, který je vytvořen na g-mailu. Po vyplnění všech potřebných informací do příslušného appenderu včetně portu, protokolu a přihlašovacích údajů jsem vytvořil testovací výjimku. Tu jsem zaslal s nejvyšší prioritou (fatal) do příslušného loggeru. V příchozí schránce e-mailového účtu, na které logy posílám, nic nezobrazilo. Problém byl v nastavení účtu, ze kterého jsem logy odesílal. Totiž v nastavení g-mailového účtu je zapotřebí nastavit, že se k tomu účtu můžu přihlásit z méně důvěryhodných aplikací. Tato možnost je v základu nastavena na vypnuto. Samozřejmě tato možnost slouží k jakési ochraně účtu, jenže to byl velký problém při testování. Po správném nastavení jsem tedy úspěšně odeslal e-mail s příslušnou hláškou nastavenou v appenderu. Ovšem jak jsem zmínil, příliš logování škodí. Pokud jsem odeslal jeden e-mail v jednu chvíli, aplikaci to nijak neovlivnilo. Jakmile jsem začal přidávat více e-mailů, už nastal problém. Aplikace se po dobu několika vteřin zastavila při zaslání pěti e-mailů naráz.

Při ukládání do souboru jsem naopak na žádný problém nenarazil. Naopak to bylo velice snadné. Loggy je možné dokonce ukládat do více souborů najednou, v různých místech. Z toho vyplývá, že je potřeba vytvořit více appenderů. Opět je zapotřebí nastavit, v jakém formátů se budou loggy ukládat a navíc lze vytvářet logy denní/týdenní a podobně dle názvu souboru. Log4net umožňuje získat aktuální čas, a tedy do souboru se bude ukládat podle aktuálního času. Tímto způsobem se vytváří časové logy. Je samozřejmostí, že loggy se nebudou přepisovat, ale budou postupně přibývat další řádky. Ovšem je tu možnost vytvoření i souboru, který se bude přepisovat. Při logování do souboru jsem došel k výsledku, který je zobrazen v následujícím bloku

```
2018-03-30 23:05:25,174 [9] INFO kresleni.Program - Test
2018-03-30 23:05:28,138 [9] WARN kresleni.Program - Data Exception.
2018-03-30 23:05:32,372 [9] INFO kresleni.Program - Test
2018-03-30 23:05:33,683 [9] WARN kresleni.Program - Data Exception.
2018-03-30 23:05:38,081 [9] INFO kresleni.Program - Test
2018-03-30 23:05:39,475 [9] WARN kresleni.Program - Data Exception.
2018-04-09 21:16:45,903 [8] ERROR kresleni.Program - Error
```

Výsledkem je přehledný a srozumitelný log, ve kterém je vidět datum a čas události, důležitost, místo události a k tomu přidaný popis. Logování do konzole vypadá ve-

lice podobně, záleží jen na nastavení šablony pro výpis.
Nejzajímavějším způsobem je logování na cloudové řešení. Log4net ve výchozím nastavení nemá žádný appender, ale některá cloudová řešení, která podporují log4net, vytvořila balíčky, které po nainstalování z nuget packages umožní přidat speciální appender do konfiguračního souboru. Na internetu je velké množství cloudových řešení, který práve log4net podporují. To je největší výhodou log4netu. Jelikož je rozšířený, umožňuje logovat na spousty cloudových řešení. Pro testovací účely jsem vybral nástroj Elmah.io, Loggr.net a Google Cloud.

Benchmarkový test nástrojů

Internetová stránka loggly prováděla benchmarkový test [6], který byl zveřejněn v roce 2016. Test se zaměřoval na pět populárních logovacích nástrojů pro .NET. Cílem testu bylo najít nejrychlejší logovací nástroj. Bylo použit testovací server, který běžel na Windows 2008 s 8GB RAM. Aplikace byla vyvíjena ve Visual Studio 2012 s .NET verzi 4.5 a primárně psaná v `C#`. V testové aplikace byla vytvořena smyčka o sto tisíci iterací, kde každou iteraci byl poslán všemi nástroji log s úrovní debug. Ve výsledku byl NLog nejrychlejším řešením, což bylo vcelku překvapivý závěr. Naopak nejpomalejším byl Elmah.
Závěrem bylo doporučeno, že pro aplikace které jsou pod velkou zátěží by bylo nejlepší vybrat NLog. Pokud by se nejednalo o takto zatěžované aplikace, nejlepší možností by byl log4net kvůli jeho jednoduché konfiguraci.

Při testování různých nástrojů uvedených v kapitole 4.2 byly dosaženy podobné výsledky, jen ne v tak velké testovací míře. V závěru bylo zjištěno, že pokud se loguje více informací najednou na více míst, dojde k zamrznutí aplikace. Hlavně při logování na e-mail. Celkově tak větší zamrznutí dojde při logování pomocí nástroje log4net. Je tedy dobrým zvykem vytvářet aplikace na bázi vláken, jelikož právě tím se vyhneme zamrznutí aplikace.

Loggr

Jako první jsem vyzkoušel Loggr.net, protože je nejjednodušší na použití. Stáhl a nainstaloval jsem potřebný balík loggr.log4net pomocí nuget balíčků. Základní konstrukce appenderu se mi ale nepřidala do log4net.configu, ale do App.config. Jediné, co bylo potřeba vyplnit byly logKey a apiKey. Tyto klíče jsem získal ze svého repozitáře na Loggr.net. Po zadání klíčů jsem ve svých aplikací vytvořil instanci logeru, zavolal na ni metodu log, do argumentu přidal potřebnou výjimku a odeslal. Vše proběhlo bez problému. Dokonce celý proces nezpůsobil „zaseknutí“ aplikace. Samotné cloudové řešení není příliš profesionální, ale pro testovací účely, či pro menší projekt velice dostačující. Celé GUI není příliš pěkné a určitě by se na něm dalo hodně věcí vylepšit. Vylepšit by se dalo například zobrazení četnosti vyjímek v grafu, rozšířit možnosti výpisu chyb a jejich detailu. Spíš mi připadá, že v určitém roce bylo cloudové řešení vytvořeno, byla nahraná verze GUI a poté se příliš neaktualizovala. Je ovšem možné vidět ve kterém dni došlo k jaké výjimce, je možné k zobrazení graf,

který ukazuje množství výjimek za určité časové období. Logy lze filtrovat, uložit pro přehlednost, zjistit o logu více informací (který uživatel log poslal, na kterém místě se vyskytl). Jediné celkové omezení je zasílání 101 logů za den.

Elmah

Dalším řešením byl Elmah.io. Stejně jako Loggr vypadal velice jednoduše, jen přicházel s velkým omezením pro mé testování. Jelikož Elmah je placené řešení, mohl jsem nástroj vyzkoušet pouze po dobu 30 dní jako trial verzi. Ta ovšem neobsahovala žádné velké omezení, kromě stejně jako u Loggru 101 logů za den. Velkou výhodou oproti Loggru bylo lepší a přehlednější GUI. Bylo vidět na první pohled, že Elmah je více propracovanějším nástrojem a je využitelný ji k větším projektů. Tedy v lepším grafickém znázornění bylo možné vidět v grafu kdy nastala logovaná chyba, postupně se chyby sčítaly, chyby bylo možné rozkliknout a vidět v jakém místě nastala a podobně. Graf byl tentokrát sloupcového typu. Bylo možné přidat více uživatelů pro logování, bylo poté vidět který uživatel loguje a více informací o něm.
Zároveň elmah disponuje mnoha výběry logovacích nástrojů, které mohou logovat události. Všechny možné technologie jsou uvedeny [5].
Další jeho velkou výhodou je odesílání e-mailů

- Při nově vzniklé chybě
- Každodenní souhrn obsahující všechny logy za posledních 24 hodin
- Pokud je error logován více často než obvykle

Díky této vlastnosti už přímo v aplikaci nemusíme použít logování do e-mailové schránky, když to obstarává sám cloudový nástroj elmah. Přehledný log na cloudové řešení Elmah je zobrazeno na následujícím obrázku, kde je vidět počet uspořádání logů na určitý den.

Obrázek 4.1: Znázornění grafu pro výpis chyb za den

Výpis nejčastějších a nedávno logovaných chyb znázorňuje Elmah tímto způsobem

RECENT	FREQUENT
err a few seconds ago	11 × Value does not fall within the expected range.
Value does not fall within the expected range. a few seconds ago	10 × nwm
err a few seconds ago	10 × ok

Obrázek 4.2: Výpis nejčastějších a nedávno logovaných chyb

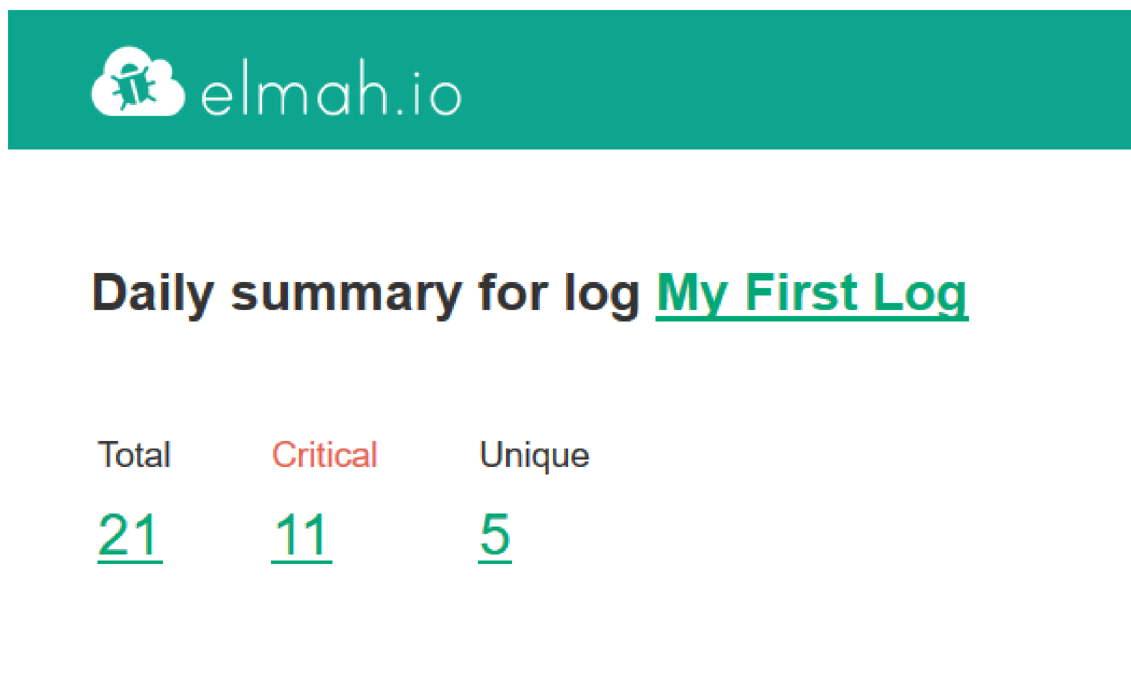
U každého logu může zobrazit více informací po rozbalení žádaného logu. Následující ukázkou je odesílání e-mailu při nově zalogované chybě, kdy nástroj zašle na nastavený e-mail daný log s popisem.

Value does not fall within the expected range.

Time	2018-05-12T20:46:46
Severity	Fatal
URL	
Method	
Status Code	
Host	DESKTOP-SOQR6SE
Error	
Application	elmahTest.vshost.exe
Source	elmahTest.Program
User	DESKTOP-SOQR6SE\Lukáš

Obrázek 4.3: Příchozí e-mail po nově vzniklé chybě

Na závěr je možné nastavit každodenní výpis celé sledované aplikace. Ve zprávě je vidět v jakém logu došlo k celkovému počtu chyb, počtu kritických chyb a unikátních chyb.



Obrázek 4.4: Každodenní výpis logu

Google Cloud - Stackdriver

Dalším nástrojem je cloudové řešení od společnosti Google - Stackdriver. Tento nástroj jsem začal testovat na začátku března, kdy jsem postupně následoval pokyny v dokumentaci. Opět bylo zapotřebí stáhnout tentokrát již dva balíčky pro logování. Dále vytvořit XML soubor pro log4net logování, a nakonec opět vytvoření instance logeru v kódu a odeslání logu. Při spuštění aplikace a zmáčknutí testovacího tlačítka se měl odeslat log na Stackdriver. Aplikace žádnou chybu nenapsala, tedy jsem se podíval do logu a nic se neobjevilo. Po pár hodinách testování jsem zjistil, že tento nástroj nejdříve ukládá logy lokálně a až potom je odesílá na Stackdriver. Tedy bylo vždy potřeba před ukončením aplikace instanci logu flushnout. Upravil jsem svůj kód a přidal flushování před ukončením aplikace. Opět jsem aplikaci spustil a opět se log nezobrazil ve Stackdriveru. Po několika dnech zkoušení a hledání jsem narazil na vzniklý thread issue v github repozitáři pro Google Cloud Logger. K mé smůle na začátku března vznikla chyba pro log4net logování právě na Google Cloud Logger. Problém byl právě flushování, jelikož pro .NET ten kód vůbec nefungoval. Issue bylo vytvořeno dne 26. února. V půlce března byla přidána odpověď, že se ví kde je chyba, a chyba by měla být opravena. Ke zlepšení nedošlo, ale přesunul jsem se

na další nástroje, které mohu otestovat. Na začátku dubna jsem se opět dostal ke Stackdriveru, a chyba stále přetrvává. Dokonce to threadu byl přidán příspěvek od dalšího uživatele, přesněji 24. dubna, který opět stejnou chybu popisuje.

Stackify - Retrace

Posledním zkoušeným nástrojem pro log4net cloud logování je nástroj od Stackify. Je to spíše nástroj pro logování aplikací, které komunikují se serverem, ale má i využití pro .NET aplikace. Největší nevýhodou je trial verze ke zkoušce pouze na 14 dní. Opět umožňuje nainstalování appenderu pomocí nuget balíčku nebo je zde možnost přímo použít jejich API. To lze opět získat pomocí nuget. Já jsem tentokrát vyzkoušel jejich API, jelikož použití jednotlivých appenderů je pořád to stejné. API k logování nabízí mnoho nástrojů a největší výhodou jejich použít je řešení situace, pokud daný nástroj nepodporuje verzi frameworku, nad kterým pracujeme. Tedy po nainstalování balíčku bylo pouze zapotřebí již ve vlastním programu vytvořit instanci logeru, zavolat logovací metodu a to bylo vše.
Retrace nabízí zobrazování logů, chyb a zároveň celkově monitoring aplikace (performance) a zároveň nabízí profiling. To se ovšem využívá u webových aplikací, tudíž mě nejvíce zajímaly logy. Po vytvoření zkušebních logů jsem musel několik desítek vteřin čekat, než se logy objevily v cloudu. Zároveň se mi aplikace výrazně pozastavila. Poté co se logy zobrazily, bylo možné vidět časovou osu (graf), ve kterém je vidět v jaký čas a den se objevily jaké chyby. Dále se samotné logy zobrazí v okně, který připomíná konzoly. Je to v ní přehledně zformátované. Lze vidět čas, datum, název logu, level logu, a lze rozkliknout pro více informací (která aplikace, umístění a podobně).

4.2.3 NLog

NLog je dosti podobný nástroj, jako log4net. Lze ho získat pomocí nuget balíčků. Stejně jako u log4net se jedná o XML schéma, které se postupně upravuje. První změna přichází u appenderů. Právě NLog jimi již nedisponuje. Struktura log4net disponovala elementy root, kde se definovaly appendery, a dále samotné použití appenderů. Zde je místo elementu root element rulers a místo appenderů jsou zde targety. Následně potom samotná definice targetů je téměř identická, jen jsou atributy elementů lépe srozumitelné, není zapotřebí jich definovat mnoho a strukuta je potom přehlednější.
Další změna přichází při vytváření instance logerů. I je NLog lepším nástrojem při mém testování nad log4netem v jednoduchosti. Totiž log4net používá ILog a NLog Logger. Při vytváření instance v log4netu je zapotřebí GetLogger(typeof(MyClass)), kdežto u NLogu je syntaxe GetCurrentClassLogger(). Zde si NLog zjistí již třídu sám. Dále dle mého názoru největší změnou je samotné logování, kdy u log4netu stačilo dát do logovacího atributu již samotnou výjimku nebo zprávu, NLog vyžaduje 2 atributy. Zprávu a výjimku. Dle mého názoru je obtížnější přejít z log4net na NLog právě z tohoto důvodu.
Poté jsem pro informaci zjišťoval, proč více vývojářů používá log4net. Zde je ovšem

důležité si uvědomit, pro jakou aplikaci si logovací nástroj vybíráme. V mých testovacích aplikací jsem došel k závěru, že NLog je rychlejší než log4net, a právě to je stěžejní důvod, proč vývojáři sáhnou po Nlogu. Naopak log4net je jednodušší na celkově použití v aplikaci. Tyto hypotézy mi byly potvrzeny, když jsem se díval na názory ostatních při různém testování.

4.3 Vyhodnocení nástrojů pro monitoring

Pro monitoring jsem vyzkoušel dva nástroje, které dle mých průzkumů a vyhodnocení vedly nad ostatními. Jsou to již zmiňované nástroje log4net a NLog.
Dle mého názoru bych při potřebě monitorovat desktopovou aplikaci vybral nástroj log4net s použitím cloudu elmah a to hlavně z důvodu jeho rozšířenosti mezi vývojáři. Jelikož je velice rozšířený a používaný, je velká šance, že pokud dojde k chybě, bude rychle objevena a opravena. Dále je díky jeho rozšířenosti podporován mnoho dalšími cloudovými řešeními, které lze využívat jako další způsob logování a dle mého názoru je to zároveň nejlepší způsob. Zároveň mimo logování na cloudové řešení podporuje logování do konzole, souboru, e-mail, databáze. Je jednoduchý na použití a v kódu je zároveň jednoduchý na údržbu.
Oproti tomu NLog má ještě více jednodušší strukturu XML dokumentu, a díky tomu se stává více přehlednám při jeho konfiguraci. Bohužel tím, že je méně používaný, mnoho cloudových řešení ho nepodporuje, avšak na druhou stranu ty velké řešení jako je například Google Cloud ho podporují. Statistiky o tom, který je z těchto nástrojů používanější beru z čísla, které je udávánu při stahování balíčku přes nuget. Dále NLog umožňuje lepší instantizaci. Zároveň je o mnoho rychlejším nástrojem než log4net. Oproti tomu je log4net celkově jednodušší na použití.

5 Výsledné zhodnocení testováných nástrojů

V kapitole 4.2.1 jsem uvedl na jakých aplikacích budou uvedené vlastnosti zkoušeny. V následujících tabulkách budou uvedeny výsledky s popisem postupu při testování. Hodnocení je uváděno na stupnici 1 až 3, kde 1 je nejlepší výsledek a 3 je nejhorší. Zároveň světle modrou barvou bude vysvícen nástroj, který byl při vyhodnocení nejlepším nástrojem.

5.1 Deployment

Velkou výhodou WiX Toolsetu je právě možnost vytvoření instalačního souboru s GUI nebo jako tichou instalaci. Pokud k hodnocení přidám i prerekvizity tak to je ten důvod, proč se WiX Toolset ukázal jako nejlepší testovací nástroj. Všechny tyto vlastnosti WiX zvládl skvěle.

Nástroj	Tiché instalace	Instalační GUI	Oprávnění	Podpisy	Prerekv.
VS Installer	2	1	1	3	3
Ad. Installer	2	2	1	3	3
WiX	1	1	1	2	1
msbuild	1	3	1	1	2

Tabulka 5.1: Hodnocení nástrojů pro deployment na základě testování

5.2 Monitoring

Dále jsou vyhodnoceny nástroje pro logování (log4net a NLog). Způsob logování znamená jakými způsoby nástroj umožňuje logy ukládat (konzole, soubor, databáze, cloud, e-mail) a připojení knihoven znamená, kolik knihoven a přídavných souborů potřebuje daný nástroj při instalaci aplikace mít u aplikace při jejím spuštěním. V tabulce není uveden počet souborů, pouze hodnocení (3 znamená značně více souborů - špatná vlastnost).

Způsob logování oba testované nástroje zvládly stejným způsobem, ale právě díky tomu, že log4net nepotřebuje tolik knihoven k spuštění aplikace a dále jejímu monitoringu, je důvod, proč log4net je lepším nástrojem. Zatížení runtimu bylo sledováno při odesílání větší počtu logů.

Nástroj	Způsob logování	Složitost konfigurace	runtimu	Připojení knihoven
log4net	1	1	3	2
NLog	1	2	2	3

Tabulka 5.2: Hodnocení nástrojů pro monitoring na základě testování

5.2.1 Cloudové řešení pro log4net

V následující tabulce jsou vyhodnoceny cloud platformy používané při logování pomocí nástroje log4net. Hodnocení je stejné, jako v předchozím případě. Tedy 1 nejlepší výsledek a 3 nejhorší. Elmah je jasně nejlepším testovacím nástrojem, ale jako náhradní řešení byl vybrán Stackify, kvůli jeho přehlednosti, zobrazování logů a více informací o nich.

Nástroj	Složitost konfig.	Přehlednost	Zasílání e-mailu	Podrobnější zobr.
Elmah	1	1	1	1
Loggr	1	2	3	2
Stackify	2	1	2	2

Tabulka 5.3: Hodnocení cloudových řešení pro log4net

6 Závěr

Na základě definovaných požadavků byly shrnuty a popsány metodiky vývoje software, dále byly popsány požadavky na deployment, monitoring, ladění a údržbu .NET aplikací. Zároveň při popisování metodik bylo zjištěno, že návrh software je nejdůležitější část z celého životního cyklu, jelikož dobře navržený software je dále dobře monitorovatelný, připravený k deploymentu a ladění.

Nejvíce zmiňovaný a používaný nástroj pro deployment je právě WiX Toolset, kvůli jeho širokým možnostem při tvorbě instalačního souboru. Zároveň v této fázi nastal největší problém celé práce, a tím je podepisování aplikací a instalačního souboru a zároveň přidávání a kontrola prerekvizit při instalaci aplikace na koncové zařízení. Jelikož WiX Toolset má nástroj na podepisování, spoléhal jsem na jeho jednoduché použití stejně jako ostatních nástrojů. Zmýlil jsem se a právě kvůli nefunkčnosti nástroje Insignia v mém testovacím případě byl použit nástroj Signtool k podepisování jednotlivých segmentů. Zároveň WiX Toolset přímo nepodporuje prerekvizity aplikací třetích stran, bylo zapotřebí prerekvizity třetích stran zakomponovat do XML souboru a dále je napojit na vytvořený bootstrapper.

Nejlepším nástrojem pro monitoring byl vyhodnocen log4net kvůli jeho jednoduché konfiguraci a jednoduché implementaci do samotné aplikace a dále jeho použití. Díky velké rozšířenosti mezi uživateli log4net podporuje spousty cloudových řešení, které jsou velice dobrým způsobem pro logování. Ve výsledku je doporučeno logovat jak na cloud, tak do textového souboru a nejvážnější výjimky odesílat e-mailem přímo správci. Jako nejlepší cloudovým řešením se ukázal Elmah. Zároveň je také velice oblíbeným v komunitě, a slouží pro více nástrojů (NLog a další). Je velice přehledný, jednoduchý na použití, lze rozbalit všechny výjimky k detailnímu popisu chyby. Také umožňuje sledování unikátních chyb. Dále umožňuje odesílání e-mailových správ jako denních souhrnů nebo nových errorů. Tím pádem nemusíme používat e-mailové logování přímo v naší aplikaci.

Při monitoringu nedošlo k žádným vážným problémům, kromě testování cloudového řešení od Googlu. Ten byl v době implementace a testování mimo službu co se týče komunikace s nástrojem log4net. Na githubu byla chyba oznámena a vývojáři cloudového řešení oznámili že na chybě pracují. Místo tohoto nástroje jsem tedy vybral cloudové řešení Loggr a Stackify, které je v celku dostatečným řešením. V porovnání doporučuji použít elmah.

Dále bylo zjištěno, že některé logování aplikace vyžaduje velký procesní čas.

V některém případě došlo k celkovému zamrznutí aplikace po dobu několika vteřin. Zde se dostáváme k případu, že logovat je potřeba s mírou. Příliš logování škodí a naopak málo logování také. Došel jsem k závěru, že pokud jsou logy posílány na e-mail, bylo by dobré aplikaci rozdělit do více vláken. Některé posílání logů na e-mail trvaly několik vteřin a právě proto byl doporučen nástroj Elmah, který zasílání e-mailu řeší ve svém cloudovém řešení. V závěrečném shrnutí by bylo nejlepší logování nastavit do vlastních vláken kvůli bezpečnosti a zároveň celkové stabilitě systému.

Jako další rozvoj práce by mohl být například zkoumání webových aplikací (deployment, monitoring ladění a údržba). Následně by mohla práce popisovat profiling jak desktopových, tak i webových aplikací s komunikací se serverem i s databázemi.

Literatura

[1] LEAU, Yu Beng, Wooi Khong LOO, Wai Yip THAM a Soo Fun TAN. Software Development Life Cycle AG ILE vs Traditio nal Approaches. 2012 International Conference on Information and Network Technology (ICINT 2012): IPCSIT vol. 37 (2012) © (2 012) IACSIT Press, Singapore [online]. 2018, 2012(37), 6 [cit. 2018-02-11]. Dostupné z: https://pdfs.semanticscholar.org/69b1/9ddc8a578f4c63d1dfe15252a465ee12fe5d.pdf

[2] MUNASSAR, Nabil Mohammed Ali a A. GOVARDHAN. A Comparison Between Five Models Of Software Engineering. International Journal of Computer Science Issues [online]. 2010, 2010(7), 7 [cit. 2018-02-12]. ISSN 1694-0814. Dostupné z: https://pdfs.semanticscholar.org/3a4a/2cb2328e2f416be0be012e5d580975943554.pdf#page=94

[3] AWAD, M. A. A Comparison between Agile and Traditional Software Development Methodologies [online]. The University of Western Australia, 2005 [cit. 2018-02-12]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.6090&rep=rep1&type=pdf. Projekt. The University of Western Australia.

[4] Deploying the .NET Framework and Applications: .NET Framework 4.6 and 4.5 [online]. [cit. 2015-10-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/6hbb4k3e(v=vs.110).aspx

[5] Elmah.io documentation. *Elmah* [online]. neuvedeno: neuvedeno, neuvedeno [cit. 2018-05-07]. Dostupné z: https://docs.elmah.io/

[6] MARSH, Jannifer. Benchmarking 5 popular .NET logging libraries. *Loggly* [online]. 2016, **1/2016**(neuvedeno), 1 [cit. 2018-05-07]. Dostupné z: https://www.loggly.com/blog/benchmarking-5-popular-net-logging-libraries/

[7] Apache log4net Manual - Introduction. *Apache* [online]. neuvedeno: Apache, neuvedeno [cit. 2018-05-07]. Dostupné z: https://logging.apache.org/log4net/release/manual/introduction.html

[8] Bootstrapping. *Firegiant* [online]. neuvedeno: .NET Foundation, neuvedeno [cit. 2018-05-07]. Dostupné z: https://www.firegiant.com/wix/tutorial/net-and-net/bootstrapping/

[9] Documentation for the WiX Toolset. *Wixtoolset* [online]. neuvedeno: .NET Foundation, neuvedeno [cit. 2018-05-07]. Dostupné z: http://wixtoolset.org/documentation/

[10] Overview of the .NET Framework. *Docs.microsoft.com* [online]. 2017, [cit. 2017-11-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview

[11] Introduction to Code Signing. *Msdn.microsoft.com* [online]. 2017 [cit. 2017-11-25]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms537361(v=vs.85).aspx

A Obsah přiloženého CD

- text bakalářské práce

  `bakalarska_prace_2018_Lukas_Vosecky.pdf`

  `kopie_zadani_bakalarska_prace_2018_Lukas_Vosecky.pdf`

- textový soubor pro vysvětlení použití skriptů a konfiguračních souborů

  `readmefirst.txt`

- skriptovací soubor pro automatizaci vytváření instalačních souborů

  `wix.bat`

- konfigurační XML soubor nástroje WiX Toolsetu

  `product.wxs`

- konfigurační XML soubor nástroje log4net

  `log4net.xml`

- testovací certifikát

  `key.pfx`

- instalační soubor aplikace

  `Product.msi`