TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Nové trendy ve vývoji software

Diplomová práce

Studijní program: N6209 – Systémové inženýrství a informatika
Studijní obor: 6209T021 – Manažerská informatika

Autor práce: **Bc. Lenka Křůmalová**
Vedoucí práce: doc. Ing. Klára Antlová, Ph.D.

Liberec 2018

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Akademický rok
2017 / 2018

Zadání diplomové práce

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Lenka Křůmalová**
Osobní číslo:	E16000449
Studijní program:	B6209 Systémové inženýrství a informatika
Studijní obor:	N6209T021 – Manažerská informatika
Zadávající katedra:	katedra informatiky
Vedoucí práce:	doc. Ing. Klára Antlová, Ph.D.
Konzultant práce:	Bc. Jan Rudolf Team leader, Neatcode, s.r.o.
Název práce:	**Nové trendy ve vývoji software**

Zásady pro vypracování:

1. Stanovení cílů a formulace výzkumných otázek.
2. Analýza technických možností využitelných při vývoji software.
3. Analýza současných požadavků v oblasti software.
4. Vývoj aplikace pomocí zvolené technologie.
5. Formulace závěrů a zhodnocení výzkumných otázek.

Seznam odborné literatury:

GOLDSTEIN, Alexis, Louis LAZARIS a Estelle WEYL. *HTML5 & CSS3 for the real world.* Second edition. Collingwood: SitePoint, 2015. ISBN 978-1-457-19278-4.

PAVLÍČEK, Antonín a Alexander GALBA. *Moderní informatika.* Praha: Professional Publishing, 2012. ISBN 978-80-7431-109-3.

SHARKIE, Craig a Andrew FISHER. *Responzivní webdesign: okamžitě.* Brno: Computer Press, 2015. ISBN 978-80-251-4384-1.

ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie.* Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.

PROQUEST. 2017. *Databáze článků ProQuest* [online]. Ann Arbor, MI, USA: ProQuest. [cit. 2017-09-28]. Dostupné z: http://knihovna.tul.cz/

Rozsah práce:	65 normostran
Forma zpracování:	tištěná / elektronická
Datum zadání práce:	31. října 2017
Datum odevzdání práce:	31. srpna 2019

Technická univerzita v Liberci, Ekonomická fakulta

L.S.

prof. Ing. Miroslav Žižka, Ph.D.
děkan Ekonomické fakulty

doc. Ing. Klára Antlová, Ph.D.
vedoucí katedry

V Liberci dne 31. října 2017

Prohlášení

Byla jsem seznámena s tím, že na mou diplomovou 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 diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma 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.

Diplomovou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

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

Datum: 4. 5. 2018

Podpis: Křůmalová

Anotace

Tato diplomová práce se zabývá především aktuálními trendy, které ovlivňují vývoj softwaru. Diplomová práce je rozdělena do dvou částí, teoretické a praktické. Cílem teoretické části je analyzovat současné trendy v oblasti vývoje softwaru. Práce se zaměří nejprve na aktuálně dostupné možnosti, které lze při vývoji využít, a poté na požadavky uživatelů a způsob, jak jim vyhovět za pomoci dostupných možností. Cílem praktické části je potom zjištěné poznatky využít při vývoji aplikace pomocí zvolené technologie, zhodnotit, zda je v souladu se současnými trendy a najít případné možnosti zlepšení.

Klíčová slova

aplikace, cloud, internet, IoT, software, trendy, vývoj

Annotation

New trends in software development

This diploma thesis is focused on the current trends that influence software development. The diploma thesis is divided into two parts, theoretical and practical. The aim of the theoretical part is to analyze current trends in software development. The diploma thesis will focus firstly on currently available options that can be used in development, then on user requirements and how to satisfy them with the help of available options. The aim of the practical part is to use the findings for evaluating an application developed with selected technology and for discovering new possible improvements.

Key Words

application, cloud, development, internet, IoT, software, trends

Obsah

Seznam zkratek **10**

Seznam tabulek **13**

Seznam obrázků **14**

Úvod **16**

1. Úvod do problematiky **17**

1.1 Literární zdroje **17**

1.2 Základní pojmy **18**

1.2.1 Data, informace, znalosti 18

1.2.2 Informatika 19

1.2.3 Informační a komunikační technologie 19

1.2.4 Počítač 20

1.2.5 Počítačové jazyky 20

1.2.6 Hardware 21

1.2.7 Software 21

1.2.8 Softwarové inženýrství 21

1.2.9 Systém 22

1.2.10 Proces 22

1.2.11 Klíčové role 23

2. Dostupné možnosti **26**

2.1 Rozvoj ICT **26**

2.1.1 Rozvoj počítačů 26

2.1.2 Internet 28

2.1.3 World Wide Web 31

2.1.4 Internet věcí 32

2.1.5 Cloud computing 33

2.1.6 Virtuální a rozšířená realita 35

2.1.7 Využití biometrik 37

2.1.8 Umělá inteligence 37

2.1.9 Virtuální měny 38

2.2 Technické prostředky vývoje **38**

2.2.1 Podpůrné softwarové nástroje 38

2.2.2 Přístupy k programování 40

2.2.3 Softwarové modelování 42

2.2.4 Volba platformy ... 49
2.2.5 Druhy aplikací ... 50
2.2.6 Programovací jazyky ... 55
2.2.7 Neprogramovací jazyky ... 58
2.2.8 Databáze ... 60
2.2.9 Business Intelligence ... 64
2.2.10 Data Mining ... 65
2.3 Řízení vývoje SW ... 66
2.3.1 Produkt ... 66
2.3.2 Projekt ... 67
2.3.3 Správa rizik ... 68
2.3.4 Řízení týmu ... 69
2.3.5 Řízení znalostí ... 70
2.4 Proces vývoje SW ... 70
2.4.1 Specifikace softwaru ... 70
2.4.2 Návrh softwaru ... 71
2.4.3 Vývoj softwaru ... 72
2.4.4 Testování softwaru ... 73
2.4.5 Provoz softwaru ... 74
2.5 Modely softwarových procesů ... 75
2.5.1 Tradiční přístup ... 76
2.5.2 Agilní přístup ... 84
2.6 Softwarová architektura ... 89
2.6.1 Architektura Klient-server ... 89
2.6.2 Vrstvená architektura ... 90
2.6.3 Architektura Master-slave ... 91
2.6.4 Architektura Broker ... 91
2.6.5 Architektura Peer-to-peer ... 92
2.6.6 Architektura Model-view-controller ... 93
2.6.7 Bezserverová architektura ... 93
3. Analýza požadavků ... 95
3.1 Softwarové požadavky ... 95
3.1.1 Specifikace požadavků ... 96
3.1.2 Inženýrství požadavků ... 97
3.1.3 Členění podle účelu ... 98

3.1.4 Členění podle vztahu k funkčnostem ... 98
3.2 Kvalita SW ... 99
3.2.1 Modely kvality SW ... 100
3.2.2 Management kvality SW ... 103
3.2.3 Kvalita kódu ... 103
3.3 User Experience ... 105
3.4 Spolehlivost softwaru ... 107
3.4.1 Bezporuchovost a dostupnost ... 109
3.4.2 Bezpečnost ... 110
3.4.3 Zabezpečení ... 111
3.5 Formy distribuce softwaru ... 112
3.5.1 Cena ... 112
3.5.2 Licence ... 113
3.5.3 Způsob ... 113
3.6 Legislativní omezení ... 113
4. Vývoj aplikace pomocí zvolené technologie ... 114
4.1 Volba technologie ... 114
4.1.1 Use-Case Driven Approach ... 114
4.1.2 Unicorn Mobile-First IoT-Ready Cloud Architecture ... 115
4.1.3 uuApp ... 116
4.2 Popis řešené problematiky ... 117
4.3 Aplikace +4U eRecept ... 119
4.3.1 Proces vývoje ... 119
4.3.2 Struktura aplikace ... 120
4.3.3 Použití aplikace ... 123
4.4 Volba hodnotících metod ... 127
4.5 Zhodnocení aplikace a procesu vývoje ... 132
Závěr ... 137
Seznam použité literatury ... 139

Seznam zkratek

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
AVAIL	Availability (dostupnost)
awid	uuAppWorkspaceId
BI	Business Intelligence
COTS	Commercial off-the-shelf
CSS	Cascade Style Sheets
CÚER	Centrální úložiště elektronických receptů
DCL	Definition Control Language (jazyk řízení dat)
DDL	Data Definition Language (jazyk definice dat)
DML	Definition Manipulation Language (jazyk pro manipulaci s daty)
ECMA	Europeann Computer Manufacturers Association)
ETL	Extract-Transform-Load
ext4	Fourth Extended Filesystem
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IA	Information Architecture
ICT	Informační a komunikační technologie
IT	Informační technologie
IxD	Interaction Design (návrh interakce)
IDE	Interactive Development Environment
JSON	JavaScript Object Notation

MR	Mixed Reality
MTTF	Mean time to failure (Průměrná doba do selhání)
MVP	Minimum Viable Product
NTFS	New Technology File System
OLAP	Online Analytical Processing
OMG	Object Management Group
OOP	Object-oriented Programming
POFOD	Probability of failure on demand (Pravděpodobnost selhání na vyžádání)
RLPO	Registr léčivých přípavků s omezením
ROCOF	Rate of occurance of failures (Frekvence výskytu selhání)
RUP	Rational Unified Process
SaaS	Software as a Service (software jako služba)
SDK	Software Development Kit
SEO	Search Engine Optimization
SGML	Standard Generalized Markup Language
SLA	Service-level Agreement (dohoda o úrovni poskytovaných služeb)
SLA	Service Oriented Architecture (architektura orientovaná na služby)
SQL	Structured Query Language
SŘBD	Systém řízení báze dat
SÚKL	Státní ústav pro kontrolu léčiv
UAF	Unicorn Application Framework
UML	Unified Modeling Language (jednotný modelovací jazyk)
UX	User Experience (uživatelský prožitek)
UxD	User Experience Design
uuBML	Unicorn Universe Business Modeling Language

uuCMD	uuCommand
VCS	Version Control System (systémy pro správu verzí)
VR	Virtual Reality
WWW	World Wide Web
XML	Extensible Markup Language

Seznam tabulek

Tabulka 1: Plus4uErecipe-main uuCommands 122
Tabulka 2: Plus4uErecipe-data uuCommands..... 123
Tabulka 3: Hodnocení aplikace +4U eRecept 132

Seznam obrázků

Obrázek 1: Klíčové role 23
Obrázek 2: Ovládání IoT s taphome 32
Obrázek 3: Využití cloudu 33
Obrázek 4: Porovnání Saas, PaaS, IaaS 34
Obrázek 5: IKEA Place 36
Obrázek 6: Třída 44
Obrázek 7: Vazby mezi třídami 45
Obrázek 8: Aktér, relace a případ užití 47
Obrázek 9: Relace mezi případy užití 48
Obrázek 10: Sekvenční diagram 49
Obrázek 11: Expense manager desktopové zobrazení 53
Obrázek 12: Expense manager mobilní zobrazení 54
Obrázek 13: Nejpoužívanější programovací jazyky Tiobe 56
Obrázek 14: Nejpoužívanější programovací jazyky StackOverflow 56
Obrázek 15: Porovnání oblíbenosti relačních a nerelačních databází 61
Obrázek 16: Data Mining 66
Obrázek 17: Postup návrhu 72
Obrázek 18: Vodopádový model 77
Obrázek 19: V-Model 78
Obrázek 20: Spirálový model 79
Obrázek 21: Model RUP 80
Obrázek 22: Pracovní toky RUP 80
Obrázek 23: Inkrementální vývoj 83
Obrázek 24: Principy agilních metodik 85
Obrázek 25: Scrum 86
Obrázek 26: Extrémní programování 88
Obrázek 27: Klient-server 90
Obrázek 28: Třívrstvá architektura 91
Obrázek 29: Master-slave 91
Obrázek 30: Architektura broker 92

Obrázek 31: Peer-to-peer 92
Obrázek 32: Model-view-controller 93
Obrázek 33: McCallův model 101
Obrázek 34: Colorblind Web Page Filter 107
Obrázek 35: Unicorn Mobile-First Iot-Ready Cloud Architecture 115
Obrázek 36: Struktura uuApp 116
Obrázek 37: Struktura aplikace +4U eRecept 121
Obrázek 38: Nový recept 125
Obrázek 39: Vystavené recepty 125
Obrázek 40: Náhled 126

Úvod

Software velmi rychle zastarává. Software vydaný před deseti lety nejspíše nebude vyhovovat současným požadavkům, je tedy třeba jej neustále rozvíjet. Jak ale zjistit, zda software současným požadavkům vyhovuje? Na tuto otázku se zaměřila práce s názvem *Nové trendy ve vývoji softwaru*.

Jak již název napovídá, cílem této práce je analýza současných trendů ve vývoji softwaru. V první kapitole jsou popsány prameny, ze kterých bylo během psaní práce čerpáno, a vysvětleny některé důležité pojmy, které se objevují v celé práci. Ve druhé kapitole se potom práce zaměřuje na možnosti, které může vývojový tým využít při vývoji aplikace. Třetí kapitola popisuje, jak zjišťovat požadavky uživatelů a jak jim vyhovět pomocí aktuálně dostupných možností. Ve čtvrté kapitole je popsán vývoj aplikace pomocí zvolené platformy. Zjištěné poznatky jsou použity pro hodnocení aplikace a procesu vývoje s ohledem na současné trendy a k nalezení možností zlepšení.

1. Úvod do problematiky

Tato kapitola by měla sloužit jako nastínění tématu, kterým se tato práce bude zabývat. V kapitole 1.1 bude popsáno, jaké zdroje literatury sloužily jako podpora při psaní práce a v kapitole 1.2 potom budou vysvětleny nejzákladnější pojmy, které je třeba znát pro čtení této práce.

1.1 Literární zdroje

V této kapitole bude popsáno, jaká literatura sloužila jako zdroj pro čerpání informací a psaní práce. Cílem bylo získat informace o aktuálních trendech ve vývoji softwaru, ale žádná z publikací toto téma nepopisuje uceleně včetně všech trendů. Proto bylo třeba rozdělit psaní teoretické části do tří bloků.

Nejprve bylo třeba zjistit, co vše vůbec spadá pod vývoj software. Pro tento účel byly velmi užitečné dvě značně rozsáhlé publikace, které jsou dosti obecné na to, aby poskytly náhled do oblasti informatiky a softwarového inženýrství. První knihou je Softwarové inženýrství od Iana Sommervilla (Sommerville 2013). Ta se zaměřuje přímo na vývoj softwaru a obsahuje mnoho cenných poznatků. Druhá z nich je kniha s názvem Podniková informatika od autorů Gály, Poura a Šedivé (Gála 2009). Ta se sice zaměřuje především na využití informačních a komunikačních technologií v podniku, ale okrajově zkoumá velké množství témat souvisejících s vývojem softwaru. Obě tyto publikace ale již nejsou úplně aktuální, proto sloužily spíše pro definování oblastí, kterými se diplomová práce bude zabývat. Další nápomocnou knihou byla Moderní informatika od Antonína Pavlíčka a Alexandra Galby (Pavlíček 2012). Ta je však velmi stručná a slouží spíše jako úvod do jednotlivých témat. Tyto publikace sloužily pro definování základní struktury práce a oblastí, kterými se práce bude zabývat podrobněji. V neposlední řadě autorka této práce získala některé znalosti problematiky během práce na pozici vývojáře webových aplikací.

Nalezené oblasti bylo potom potřeba prozkoumat hlouběji, aby bylo možné hledat aktuální trendy. K tomu už sloužily méně obecné publikace a články. Je na místě zde zmínit pět publikací, které sloužily k lepšímu pochopení problematiky a způsobu jejího využití při

vývoji aplikace. Přehled jednotlivých metodik použitelných při vývoji softwaru poskytla kniha Agilní programování (Kadlec 2004). Další užitečnou knihou byla Kvalita softwaru (Roudenský, 2016), která je i celkem aktuální a poskytuje ucelený přehled možností měření a zajištění kvality softwaru. Náhled do řízení vývoje softwaru poskytla kniha Information Systems Project Management (Avison 2009), je ale spíše zaměřena na vývoj komplexních informačních systémů. Fenomén cloud computingu prakticky popisuje kniha Cloud computing (Velte 2011), která se zaměřuje i na budoucí vývoj cloudu, přesto však nezahrnuje nejnovější trendy v oblasti cloudových služeb. Pro lepší pochopení pojmu požadavek sloužila kniha Požadavky na software (Wiegers 2008). Potom bylo samozřejmě použito mnoho dalších zdrojů pro ověření informací v jednotlivých publikacích a pro zjištění dalších podrobností. Tyto zdroje jsou vždy uvedeny u daného odstavce.

Nakonec bylo třeba ve zvolených oblastech zjistit, jaké jsou současné trendy a jak je při vývoji softwaru využít. Většinou bylo čerpáno z různých článků, doporučení a statistik, které se týkaly přímo konkrétní oblasti. Nemá smysl tu vypisovat všechny zdroje, budou zmíněné u jednotlivých kapitol.

1.2 Základní pojmy

V této kapitole se diplomová práce zaměří na některé základní pojmy, které je třeba úvodem vysvětlit. Jedná se o klíčové pojmy, které se objevují napříč celou prací, a jejich vysvětlení již na začátku umožní zaměřit se v dalších kapitolách práce přímo na konkrétní zkoumané problémy.

1.2.1 Data, informace, znalosti

Údaj popisuje určitý fakt o nějakém objektu reálného světa (Gála 2009).

Data reprezentují kolekce údajů vhodné pro počítačové zpracování. Lze je rozdělit na data *strukturovaná* a *nestrukturovaná*. (Černý 2017)

Metadata je označení pro speciální typ dat, která slouží k popisu dat jiných (Pavlíček 2012).

Informace vzniká, pokud je datům přiřazen kontext a smysl. Ten příjemce informaci přiděluje na základě svých znalostí. Proto to, co je pro někoho informací, je pro někoho jiného pouhá sekvence znaků. Informace nějakým způsobem snižuje úroveň neznalosti. (Černý 2017) Na každou informaci je možné aplikovat tři různé úrovně pohledu (Gála 2009):

- Úroveň *syntaxe*, která se zabývá vnitřní strukturou zprávy složené ze znaků abecedy.
- Úroveň *sémantiky*, která zkoumá význam jednotlivých prvků, ze kterých je informace složena.
- Úroveň *pragmatiky*, která zkoumá možnosti využití informace a její vztah k příjemci.

Znalost znamená schopnost dávat informace do souvislostí a prakticky je využívat.

Nejvyšší úrovní vědění a poznání je potom *moudrost*. Znamená schopnost vidět souvislosti a vztahy mezi entitami reálného světa za pomoci celoživotního získávání zkušeností. (Černý 2017)

1.2.2 Informatika

Informatika je vědní obor, který se zabývá zpracováním informací. Zkoumá jejich vyjádření, podobu a přenos v přirozených i umělých systémech. Informatika využívá poznatků z jiných věd, především z přírodních věd a počítačové vědy. Pojem informatika bývá často spojován s přídavnými jmény, které označují spojení informatiky s jinými vědami či její použití v určité oblasti, například manažerská informatika, neuroinformatika, podniková informatika či zdravotnická informatika. (Gála 2009)

1.2.3 Informační a komunikační technologie

Informační a komunikační technologie (Information Communication Technologies, zkráceně ICT) zahrnují širokou škálu technických prostředků a programového vybavení (Gála 2009), které slouží ke zpracování, uchovávání a přenosu dat a informací. (Potáček 2014) Rozšiřuje původní termín informační technologie, který pochází z doby, kdy mezi sebou jednotlivá zařízení nedokázala komunikovat.

Informační a komunikační technologie tedy lze rozdělit na technické prostředky, které jsou nazývány hardwarem, a programové vybavení, kterému se říká software. Oba tyto pojmy budou v následujících podkapitolách blíže vysvětleny. Největší součást informačních a komunikačních technologií tvoří počítače, jejich součásti a programové vybavení a související periférie. (Gála 2009)

1.2.4 Počítač

Technicky je počítač jakékoliv programovatelné zařízení. To znamená, že dokáže vykonat naprogramovanou sadu instrukcí a zároveň dokáže přijímat nově zadané instrukce. (Christensson 2006) V současnosti je tento pojem často využíván jako označení stolních počítačů, laptopů, či notebooků. Pro potřeby této práce bude počítač chápan jako jakékoliv zařízení, které dokáže nějakým způsobem zpracovávat informace, ať už stolní počítač, chytrý telefon, či palubní počítač v chytrém automobilu. Jednotlivé druhy počítačových zařízení budou popsány v kapitole 2.1.1 této práce. Počítače je možné rozdělit na osobní (Personal Computer, zkráceně PC), a služební, což jsou obvykle servery (Gála 2009).

1.2.5 Počítačové jazyky

Počítačové jazyky zahrnují jakékoliv typy jazyka, kterému dokáže (alespoň po zkompilování či interpretaci) porozumět počítač. Tato práce se zaměří na programovací jazyky v kapitole 2.2.6, na některé neprogramovací jazyky v kapitole 2.2.7 a potom v kapitole 2.2.8 bude zmíněn dotazovací jazyk pro přístup k databázi. V kapitole 2.2.3 bude popsán modelovací jazyk, který však není počítačovým jazykem jako takovým, je ale možné z něj do určité míry zdrojový kód generovat.

Počítačové jazyky obsahují tři základní části:

- *Abeceda* je reprezentována množinou symbolů, které je možné použít.
- *Syntaktická pravidla* definují způsob konstrukce jednotlivých výrazů.
- *Sémantická pravidla* přiřazují význam jednotlivým symbolům a výrazům.

1.2.6 Hardware

Hardware je označení pro technické prostředky sloužící k přenosu a zpracování informací. Zahrnuje počítače a jejich fyzické vybavení, periferní zařízení, technické prostředky komunikačních sítí (např. modemy), kancelářská zařízení (tiskárny, skenery), ale i specializovaná technická zařízení. (Pavlíček 2012)

1.2.7 Software

Software je programové vybavení počítače a zahrnuje počítačové programy, související dokumentaci a nástroje. (Pavlíček 2012) Místo označení software se často využívá také pojem softwarový systém. Software lze dále dělit na systémový a aplikační. Systémový software, někdy také nazývaný jako základní softwarové vybavení (Gála 2009), je obvykle operační systém a zajišťuje komunikaci s hardware. Díky němu je možné spuštění aplikačního softwaru, programu, který umožňuje provádět nějakou užitečnou činnost. (Vlach 2018) Aplikační software bývá často označován aplikací a toto označení se využívá i v této práci. Dále pod pojem software spadají programové prostředky pro podporu vývoje softwaru.

Tato práce se při popisu vývoje zaměřuje především na jednoduchý aplikační software s jedním konkrétním účelem určený pro jednotlivce či menší firmy.

1.2.8 Softwarové inženýrství

Softwarové inženýrství je technická disciplína, která se zabývá všemi apekty vývoje software. Oproti informatice, která se zabývá především teoriemi, na nichž je založeno fungování počítačů, se softwarové inženýrství zabývá praktickými otázkami vývoje. Zahrnuje čtyři základní aktivity: specifikaci softwaru, vývoj softwaru, validaci softwaru a evoluci softwaru. (Sommerville 2013)

1.2.9 Systém

Pojem systém se často využívá nejen v oblasti informatiky a informačních technologií. Jedná se o účelově definovanou množinu prvků a vazeb mezi nimi. Vlastnosti prvků a vazeb mezi nimi určují vlastnosti a chování celého systému. (Gála 2009)

Informační systém zajišťuje vhodné vyjádření informací, jejich zpracování a přenos. Je tvořen lidmi, nástroji a metodami pro zpracování informací, datovými úložišti, použitým hardwarem a procesy, které transformují vstupy na výstupy. (Avison 2009)

Operační systém je množina základního softwarového vybavení počítače, které řídí všechny ostatní programy zpracovávané počítačem. Zajišťuje komunikaci mezi vnitřním i periferním hardwarem počítače i s jinými počítači v síti a spravuje technické prostředky. Umožňuje přístup k datům uloženým na datovém úložišti prostřednictvím nějakého souborového systému (anglicky označovaného jako filesystem) a poskytuje další užitečné funkce, které mohou využít nejen uživatelé, ale také jiné programy. (Pavlíček 2012)

Souborový systém definuje členění paměťových médií na svazky, které se dále člení na sektory s definovanou velikostí. Soubor je potom uložen na jednom či více sektorech, pokud nezaplní celý sektor, zbývající místo již nemůže využít jiný soubor. Souborových systémů existuje mnoho a jsou vždy určeny pro konkrétní operační systém. Operační systém obvykle umožňuje použití několika různých souborových systémů. (Gála 2009) Operační systém Windows 10 například jako výchozí souborový systém používá NTFS (New Technology File System), oproti tomu unixové operační systémy, například Fedora, využívají souborový systém ext4 (Fourth Extended Filesystem).

1.2.10 Proces

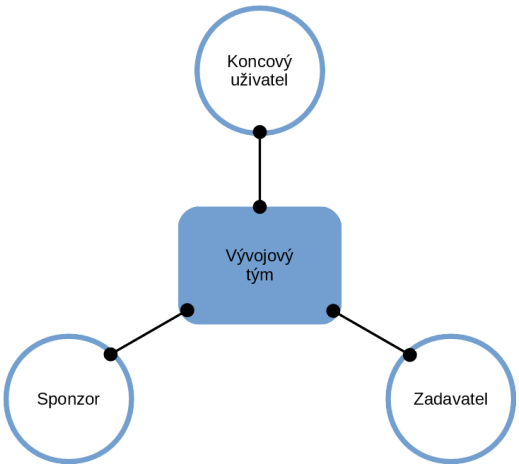
Proces je soubor vzájemně souvisejících činností prováděných za účelem dosažení určitého cíle, které transformují vstupy na výstupy. Každý proces obvykle spotřebovává nějaké zdroje, ať už lidské, materiální či jiné. (Roudenský 2016) Procesy lze rozdělit do tří skupin: (Gála 2009)

- *Základní* procesy jsou takové procesy, kterými jsou zajišťovány hlavní cíle firmy. Základní procesy vedou k přidávání hodnoty.
- *Podpůrné* procesy samy o sobě hodnotu netvoří, ale slouží jako podpora procesům, které ano.
- *Řídící* procesy potom slouží k řízení, správě a administrativním účelům.

1.2.11 Klíčové role

Tato práce se na vývoj softwaru zaměřuje spíše z technického hlediska, ale žádný vývojový proces se neobejde bez lidského faktoru. Z toho důvodu budou v této podkapitole zkoumány čtyři klíčové role, které autorka považujde za základní v oblasti vývoje softwaru.

Do vzniku nového software je obvykle zapojeno velké množství osob z různých oborů. Tato práce se bude zabývat především čtyřmi hlavními rolemi, které vývoj aplikace ovlivňují nejvíce: zadavateli, sponzory, koncovými uživateli a vývojovým týmem. Vývojový tým je středobodem celého vývoje a má za úkol zajistit komunikaci mezi jednotlivými subjekty, kterých se vývoj SW týká. Tuto skutečnost zachycuje Obrázek 1.



Obrázek 1: Klíčové role
Zdroj: vlastní

Jeden subjekt (fyzická či právnická osoba, případně skupina osob) může zastávat i více rolí najednou a často tomu tak i je. Ne všechny role se musí na vývoji software přímo podílet. Potřeby jednotlivých rolí jsou odlišné a často jdou proti sobě, je ale důležité analyzovat

potřeby každé z nich. Vývoj softwaru za situace, kdy je jedna či více stran nespokojená, je dlouhodobě neudržitelný.

Základní role použité v této práci volně vyplývají z teorie řízení projektu, (Vlach 2012) která rovněž definuje roli zadavatele a sponzora. Projektovým týmem je potom míněn vývojový tým. Ten by měl mít určitou strukturu podobnou projektovému týmu včetně rozdělení zodpovědností za každou oblast vývoje. Koncoví uživatelé jsou v teorii projektového řízení často definováni jako zákazníci či klienti.

V následujících podkapitolách bude každá z uvedených rolí konkrétně definována pro potřeby této práce.

Zadavatelé

Zadavatel určitým způsobem iniciuje vývoj aplikace. Na úplném začátku existuje pouze určitý problém, který je potřeba vyřešit. Zadavatelem je subjekt, který věří, že daný problém může být vyřešen právě vývojem nové aplikace, a má o takovou aplikaci zájem. Zadavatel by měl mít jasnou představu o tom, co potřebuje, jakými funkcemi má finální produkt disponovat, jak má aplikace vypadat. Nemá smysl vyvíjet aplikaci bez zadavatele, tedy takovou, u které neexistuje nikdo, kdo by věřil, že její vznik přinese určitou přidanou hodnotu. Požadavky zadavatelů je třeba co nejlépe definovat již na začátku vývoje.

Zadavatel aplikace obvykle vývoj také financuje a potom je zároveň sponzorem. Rovněž často bývá koncovým uživatelem. Je důležité upozornit, že zadavatel nemusí vždy být vlastníkem problému, u kterého věří, že jej vývoj aplikace může vyřešit.

Sponzoři

Sponzoři jsou neméně důležitou rolí, protože vývoj softwaru financují. Je důležité s nimi při vývoji konzultovat především finanční stránku vývoje, tedy rozpočet, a samozřejmě také neočekávané události, například pokud vývojový tým narazí na technickou překážku při vývoji, která přinese dodatečné náklady.

Sponzor obvykle zastává i některou z dalších rolí, ale ne vždy tomu tak musí být. Nejčastěji je zadavatelem, který má zájem o určitou aplikaci a je ochoten za to zaplatit. Subjekt, který zastává obě role současně, bývá často označován také jako zákazník, někdy

dokonce aplikaci sám využívá. Sponzor se v průběhu životního cyklu vývoje softwaru může měnit, může jím být například nejprve vývojový tým, poté koncoví uživatelé.

Koncoví uživatelé

Koncový uživatel je ten, kdo bude hotovou aplikaci využívat. Na vývoji se vůbec nemusí podílet. V takovém případě vývojový tým vyvíjí tzv. Commercial off-the-shelf, zkráceně COTS (ManagementMania 2016), což je označení pro softwarové produkty nabízené širokému okruhu cílových uživatelů, kteří však nemají na finální softwarový produkt žádný vliv, nepodílejí se na vývoji a pokud chtějí software používat, musí ho použít takový, jaký je. Často je uživatel a zadavatel chápán jako jeden subjekt (především v teorii projektového řízení), někdy tomu tak opravdu je, ale nemusí to tak být. Zadavatel je hnacím motorem vývoje, právě on věří, že vývoj aplikace bude mít určitý přínos, přičemž uživatel o vývoji aplikace ani nemusí vědět, je však důležité, že ve finále ji bude využívat. Proto je důležité při návrhu aplikace zkoumat potřeby nejen zadavatele, ale především koncových uživatelů, a to i když se na vývoji přímo nepodílejí. Pokud je zadavatelem někdo, kdo se v oblasti ICT moc nepohybuje, je třeba, aby vývojový tým znal aktuální trendy a doporučil zadavateli postupy klíčové pro uživatelskou přívětivost aplikace.

Vývojový tým

Vývojový tým lze považovat za nejdůležitější článek při vývoji software. Jedná se o tým sestavený z ICT specialistů, který má na starosti vývoj aplikace v průběhu celého životního cyklu. Každý vývojový tým musí mít nějakého manažera či team leadera, který je za průběh a organizaci vývoje zodpovědný. V této práci není zkoumán software, který si vyvíjejí jednotlivci sami pro sebe, je tedy vhodné zdůraznit, že vývojový tým bude vždy skupina osob, nikoli jednotlivec.

2. Dostupné možnosti

V této kapitole se práce zaměří na aktuálně dostupné možnosti, které lze využít při vývoji software. Zkoumány budou především technické možnosti, kterými je vývojový tým při vývoji omezen. V této kapitole budou také popsány některé procesy a postupy, které souvisí s vývojem v některé fázi jeho životního cyklu. Stručně zhodnoceny budou také výhody a nevýhody různých řešení.

2.1 Rozvoj ICT

Jedním z nejdůležitějších aspektů, které ovlivňují vývoj softwaru, je úroveň rozvoje informačních a komunikačních technologií. Technologie se vyvíjí poměrně rychle, a proto je třeba při vývoji využívat nových možností, které se v této oblasti objevují. V této podkapitole bude popsáno několik trendů v oblasti ICT, které je v současnosti možné využít pro vývoj aplikací.

2.1.1 Rozvoj počítačů

Jak již bylo zmíněno v kapitole 1.2.7, tato práce se zaměřuje především na jednoduché aplikace zaměřené na jeden konkrétní problém, které jsou určeny pro jednotlivce či menší firmy. Tito uživatelé budou pravděpodobně k přístupu k aplikaci chtít využívat některý typ osobních počítačů.

Počítače a jejich výkon se neustále vyvíjejí. V této souvislosti je často zmiňován tzv. Mooreův zákon. Ten byl definován v roce 1965 Gordonem Moorem a říká, že počet tranzistorů v procesorech se každoročně zdvojnásobí. (Pavlíček 2012) Praxe sice ukázala, že vývoj postupně zpomaluje, přesto je tento zákon i po více než padesáti letech uznáván. V současnosti se však hovoří o konci Mooreova zákona, další zvyšování výkonu už nemá takový smysl, vzhledem k tomu, že se pro složité úlohy využívají distribuované systémy složené z velkého množství počítačů. Vědci se snaží předpovědět, jakým způsobem se technologie bude dále vyvíjet. (Dovey 2018) Dosáhlo již lidstvo maxima v oblasti rozvoje počítačů?

Počítače lze rozdělit na nepřenosná (desktopová), přenosná a mobilní zařízení. Hlavním rozdílem je, že mobilní zařízení jsou uzpůsobena ke snadné manipulaci i během používání, ideálně ovladatelná pomocí jedné ruky, přenosná zařízení je možné přenášet, ale předpokládá se, že během samotné práce s nimi manipulováno nebude. Desktopová zařízení jsou určena k používání doma či v kanceláři bez nutnosti zařízení kamkoliv přesouvat.

Tato práce se zaměří na vývoj software pro nejčastěji používaná zařízení - chytré telefony, tablety, notebooky a stolní počítače. Ale existují také například chytré televize, hodinky či brýle, dokonce i chytré spotřebiče a další prvky domácnosti, automobily a další zařízení. Je třeba se zamyslet nad tím, z jakých zařízení by cíloví uživatelé chtěli k aplikaci přistupovat.

V současné době je nemyslitelné zapomenout při vývoji aplikace na mobilní zařízení. Jejich obliba a množství uživatelů se každoročně zvyšuje a od roku 2015 počet uživatelů mobilních zařízení převyšuje počet uživatelů desktopových počítačů. (Lella 2015) V roce 2015 mělo v České republice mobilní telefon 98% domácností a jen 75,6% domácností počítač. (ČSÚ 2016)

Volba cílových zařízení úzce souvisí s tím, jaký druh aplikace je vyvíjen. V případě nativních aplikací je důležité zaměřit se na volbu podporovaných operačních systémů, u webových aplikací na volbu podporovaných prohlížečů (což ale s operačními systémy také souvisí, ne každý prohlížeč je kompatibilní se všemi typy operačních systémů). V kapitole 2.2.5 se tato práce postupně zaměří na jednotlivé druhy aplikací.

V následující podkapitole bude stručně popsáno využití poznatků kvantové mechaniky v oblasti počítačů, které jsou zatím ale spíše ve fázi experimentů.

Kvantové počítače

Kvantový počítač je zařízení pro vykonávání výpočtů. Je založen na kvantové mechanice a uplatňuje některé její principy. Jedná se o super rychlý stroj, který dokáže provádět výpočty nad obrovským množstvím dat. Svou konstrukcí příliš nepřipomíná počítač, který byl popsán výše, podobnější je prvním analogovým počítačům než těm současným.

Základem kvantových počítačů je tzv. qubit. Jedná se o alternativu bitu v klasických počítačích. Co to je qubit, je možné přiblížit pomocí sady dvou protonů a jednoho elektronu. Předpokládáme, že jeden proton představuje hodnotu nula a druhý proton představuje hodnotu jedna. Dále jsou k dispozici dva lasery, každý pro jeden proton. Pokud na jeden z protonů posvítíme laserem, dojde k vybuzení elektronu, který odskočí k druhému protonu. Hlavní problém ale nastane, pokud jsou oba lasery spuštěny současně a stejnou silou. Potom není možné říci, zda je elektron v pozici nula nebo v pozici jedna. Elektron je totiž v obou pozicích současně, což je označováno jako superpozice. Kvantový svět je postaven na myšlence, že dokud neprovedeme přesné měření, ztrácí se pojem polohy v neurčitosti. (Mentzl 2016)

Kvantové počítače jsou velmi zajímavou oblastí informatiky a jejich pochopení vyžaduje dlouhé studium, proto není cílem této práce vysvětlit jejich fungování dopodrobna. Je však na místě je alespoň zmínit v práci, která se věnuje trendům v oblasti vývoje softwaru. Je totiž možné, že budou úzce souviset s budoucností dalšího vývoje technologií. Hlavním principem je již zmiňovaný qubit, který umožňuje nové i zatím neobjevené možnosti zpracování výpočtů. Samotná implementace kvantových počítačů je v začátcích, není možné stavět obrovské stroje obsahující miliony qubitů, jejich množství se zatím pohybuje spíše v řádu jednotek či desítek. IBM v roce 2016 spustila cloudovou službu Q, která umožnila kvantové výpočty na pěti qubitech. V minulém roce se jejich počet rozrostl na 20. (Marr 2018)

2.1.2 Internet

Komunikační prostředky slouží ke vzájemnému propojení jednotlivých počítačů do sítě, která umožňuje jejich vzájemnou komunikaci. Postupné propojování počítačů umožnilo vznik globální sítě *internet*. Aby spolu mohla komunikovat i zařízení od různých výrobců, byly definovány standardy komunikace, označované jako *protokoly*. (Gála, 2009) Jedním z nejznámnějších protokolů je Hypertext Transfer Protocol (zkráceně HTTP), případně jeho zabezpečená verze Hypertext Transfer Protocol Secure (zkráceně HTTPS), který se používá ke komunikaci v rámci služby World Wide Web, která je blíže popsána v kapitole 2.1.3.

Internet je jedním z hlavních aspektů, které ovlivňují vývoj software. Rozvoj internetu měl na vývoj software obrovský dopad a také současné internetové trendy se do vývoje promítají. Tato kapitola se postupně zaměří na výhody i nevýhody, které přináší využití internetového připojení při různých fázích vývoje.

V souvislosti s internetem se často používají dva pojmy: online a offline, a je na místě je pro úplnost stručně vysvětlit. Pojem online nese význam "připojený k internetu", offline je opakem, tedy "bez internetového připojení".

Internet zvýšil dostupnost softwarových služeb a vedl k důležitým pokrokům v oblasti programovacích jazyků. (Sommerville 2013)

Výhody využití internetu

V brzkých fázích vývoje (analýza a návrh) internet slouží především k usnadnění a urychlení komunikace, prostřednictvím internetu je možné odesílat výstupy zadavateli, který je může poupravit a poslat zpátky. Díky internetu může mít sponzor neustálý přehled o vyčerpané části rozpočtu a také o tom, na co byly peníze vynaloženy. Ve vývojovém týmu manažer konzultuje se zkušenými vývojáři možnosti realizace požadavků zadavatele. Informace ohledně vývoje aplikace mohou být během zlomku vteřiny předány jiné osobě, která se na vývoji podílí. Již v této fázi je možné zahájit internetovou marketingovou kampaň a hledat cílové uživatele pro budoucí aplikaci. Hlavní výhodou je tedy sdílení informací mezi aktéry, kterých se vývoj software týká. (Wasserman 2011)

Během vývoje aplikace může internet kromě komunikace pomoci i při týmové práci ve vývojovém týmu. Existuje mnoho nástrojů, které slouží ke sdílení zdrojového kódu v rámci týmu prostřednictvím internetu a k uchovávání historie změn. Takovým nástrojům se říká také verzovací systémy a podrobněji budou popsány v kapitole 2.2.1.

Při provozu aplikace může být internet využíván také jako součást aplikace. Je třeba zvolit vhodný druh aplikace, rozhodnout, jakým způsobem bude aplikace na internet připojena, co se bude prostřednictvím internetu odesílat či přijímat, jak budou data zabezpečena a mnoho dalších souvisejících aspektů.

Většina moderního software využívá internetového připojení, některé aplikace bez něj nemohou fungovat. Webové aplikace jsou dostupné pouze prostřednictvím internetového prohlížeče (podrobněji budou popsány v kapitole 2.2.5). Existují však případy, kdy je využití internetu komplikované (například při použití map se člověk může dostat do míst, kde internet není dostupný), potom je vhodné, aby daný software disponoval i offline verzí, která může využít data stažená v době, kdy bylo dané zařízení k internetu připojeno. Kromě toho lze prostřednictvím internetu komunikovat s veřejností a získávat nové zákazníky a zpětnou vazbu od těch existujících. Software je také často distribuován prostřednictvím internetu. Více o formách distribuce software je popsáno v kapitole 3.5. (Wasserman 2011)

Nevýhody využití internetu

Jak bylo zmíněno v předchozí kapitole, mezi hlavní výhody patří usnadnění komunikace mezi jednotlivými aktéry. Komunikace by se ale neměla omezit jen na online prostředí. Může totiž dojít ke ztrátě osobního kontaktu.

Další nevýhodou mohou být výjimečně dodatečné náklady, především při přidání online funkcí do stávající verze (příkladem může být jednoduchý editor obrázků, ke kterému chceme přidat funkčnosti pro sdílení obrázků prostřednictvím e-mailu či sociálních sítí). Pokud se s napojením na internet počítá již od začátku, může to naopak spoustu práce ušetřit. (Wasserman 2011)

Pokud se aplikace na připojení k internetu spoléhá, mohou nastat problémy při nedostatečném připojení, výjimečně může dojít i ke ztrátě dat. Tato rizika mohou být eliminována zálohováním dat.

Přibližně čtvrtina domácností v České republice internet nemá. Je třeba zhodnotit, zda budou mít cíloví uživatelé přístup k internetu, především pokud patří do nižších příjmových skupin. Vybavenost domácností České republiky internetem je pod evropským průměrem. (ČSÚ 2016)

Poslední nevýhodou je větší zranitelnost. Každá aplikace musí zohledňovat rizika (o nich více v kapitole 2.3.3), samozřejmě to platí i pro offline aplikace, ale u online aplikací je

těch rizik mnohem více. Zabezpečení aplikace před těmito riziky opět vede k dodatečným nákladům.

2.1.3 World Wide Web

Nejznámnější službou, která využívá internetovou síť, je World Wide Web (zkráceně WWW). Uživatelé pomocí software označovaného jako webový prohlížeč přistupují k datům uloženým na webových serverech. Jak již bylo zmíněno v předchozí podkapitole, pro komunikaci se využívá protokol HTTP, případně HTTPS pro případy, kdy je nutné vyšší zabezpečení komunikace. Data se přenášejí ve formě webových stránek. Webová stránka je dokument definovaný pomocí značkovacího jazyka HTML, který bude podrobněji popsán v kapitole 2.2.7. (Pavlíček 2012)

Webové stránky je možné dělit do dvou skupin – statické a dynamické. Statické webové stránky se objevovaly spíše v začátcích internetu. Od té doby však webové prohlížeče zaznamenaly obrovský pokrok. (Sommerville 2013) Nyní je většina webových stránek dynamická. To znamená, že dokáže svůj obsah měnit na základě interakce s uživatelem. To umožnilo rozvoj webových aplikací a také dalších druhů aplikací, které jsou na těch webových založeny a budou popsány v kapitole 2.2.5. (Morrow 2014)

Web 2.0, Web 3.0

V souvislosti s rozvojem webu se využívají termíny Web 2.0 a Web 3.0. Web 2.0 je pojem, který se používá od roku 2004 jako označení etapy rozvoje webu. Je založen především na myšlence, že obsah webu tvoří lidé. Web 3.0 označuje jeho další rozvoj, který zahrnuje využití umělé inteligence a přizpůsobení obsahu uživatelům. (Morrow, 2014)

SEO

Hlavním prostředkem, díky kterému se mohou uživatelé orientovat napříč webem, je webový prohlížeč. Aby uživatelé webovou stránku či aplikaci našli, musí se objevit na prvních příčkách ve výsledcích vyhledávání při zadání souvisejících klíčových slov. Search Engine Optimalization (zkráceně SEO) je označení pro přístup, který spočívá v budování takových stránek, které dokážou prohlížeče automaticky zpracovat, s cílem umístit stránku na první příčky výsledků vyhledávání. (Palkovič 2017)

2.1.4 Internet věcí

V současné době se do internetové sítě již nezapojují jen počítače a mobilní zařízení. Cílem internetu věcí (Internet of Things zkráceně IoT) je propojit prostřednictvím internetu nejrůznější zařízení z reálného světa. Jedná se o různé spotřebiče, automobily, prvky domácnosti, nositelná zařízení (například hodinky) a další. Tato zařízení obsahují obvykle nějaká čidla a senzory, pomocí kterých vyhodnocují realitu a podle toho dále postupují. Umožňují odesílat majiteli notifikace či se samostatně rozhodovat, to už závisí na úrovni umělé inteligence daného zařízení. Zařízení jsou propojována za účelem snazší vzájemné komunikace, získávání a analyzování dat, hromadného ovládání několika různých zařízení i jejich automatického fungování na základě vyhodnocování okolností. (Morrow 2014)

Odhaduje se, že v roce 2020 bude internet věcí zahrnovat 30 miliard zařízení (Statista 2016). Hranice nejsou vymezeny, takže chytrým (smart) zařízením se může stát cokoliv, u čeho to má alespoň minimální smysl. Při vývoji softwaru je vhodné se zamyslet nad tím, jaká zařízení by mohla s aplikací komunikovat a jak toho dosáhnout. Jejich využití může poskytnout cenná data, usnadnit život osob a přináší mnoho dalších výhod. Je třeba však myslet na to, že zapojení zařízení do sítě internet i komunikace s takovým zařízením v rámci aplikace vždy přináší další rizika.

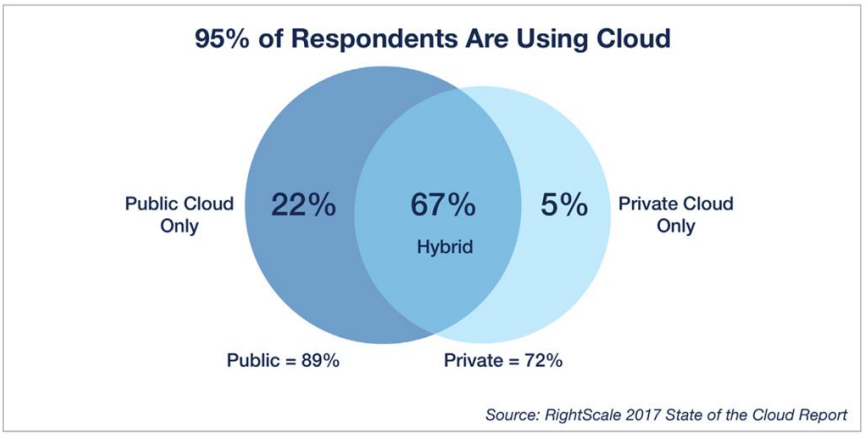
Na obrázku níže je ukázka aplikace taphome (ATIcomp 2018), která umožňuje vzdálenou správu domácnosti (samozřejmě vyžaduje využití specifických zařízení, senzorů atd., které je třeba zakoupit).

Obrázek 2: Ovládání IoT s taphome
Zdroj: ATIcomp 2018

2.1.5 Cloud computing

Cloud computing úzce souvisí s využitím internetu. Jedná se o dost abstraktní pojem, který zahrnuje většinu služeb dostupných přes internet i technologie, které využívají. Označení cloud vzniklo ze skutečnosti, že internet se často v různých schématech a diagramech označoval symbolem mraku. Zahrnuje servery, úložiště, databáze, sítě, sotware, analytické nástroje a další. Patří sem tedy téměř vše, co je provozováno v rámci sítě internet. (Velte 2011) Společnosti, které tyto služby nabízejí, jsou označovány jako poskytovatelé cloudu a jejich zákazníci za službu platí v pravidelných intervalech podle využití. Mezi nejúspěšnější poskytovatele cloudu v současnosti patří Microsoft, Amazon a IBM. (Evans 2017)

V současnosti je velmi oblíbený tzv. hybridní cloud, který respektuje, že firmy citlivá data nechtějí, případně ani nemohou zpřístupnit externím poskytovatelům. Hybridní cloud je výpočetní prostředí, které kombinuje privátní (firemní) a veřejný cloud a umožní škálovat místní infrastrukturu do veřejného cloudu pro zpracování nadbytečné poptávky. (Microsoft 2018a) Průzkum společnosti RightScale odhalil, že cloudové služby využívá 95% respondentů a 67% respondentů využívá hybridní cloud. Obrázek 3 tuto skutečnost znázorňuje pomocí grafu.



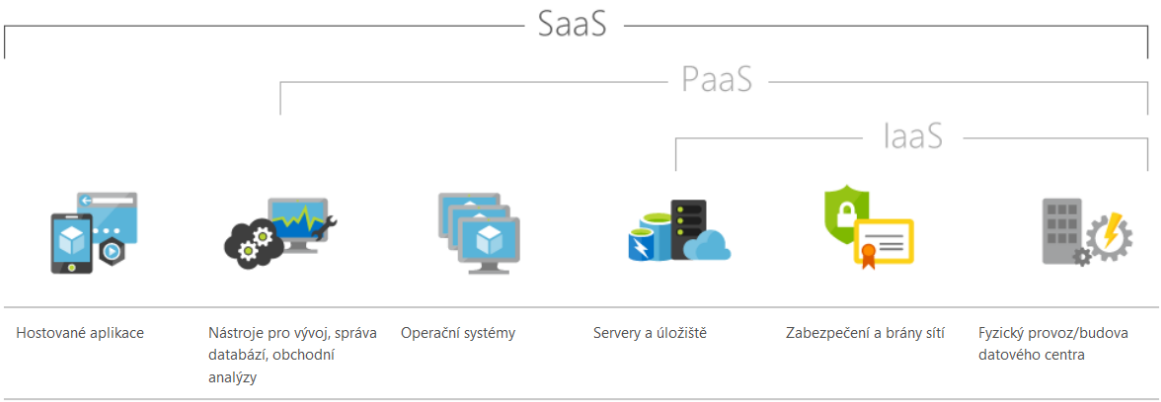
Obrázek 3: Využití cloudu
Zdroj: RightScale 2017

Využití cloudu přináší mnoho výhod. Hlavní výhodou je, že zákazník platí jen za to, co skutečně využije. Cloudové aplikace běží na serveru, který výkon poskytuje na vyžádání. Když je potřeba, dokáže během několika vteřin zpracovat i náročné výpočetní úlohy a ve

zbylém čase neblokuje zdroje zbytečně. K aplikaci i jejím datům je možné přistupovat odkudkoliv. Data jsou pravidelně zálohována a stejně tak existují i záložní servery, na kterých mohou cloudové služby běžet, pokud některý ze serverů přestane fungovat. (Velte 2011)

Cloud computing s sebou samozřejmě přináší i nevýhody. Jeho největší nevýhodou je nedostupnost služeb v případě výpadku internetu. Mimořádně může dojít i k nedostupnosti cloudových služeb, přestože poskytovatelé využívají různé prostředky, aby tomu zabránili. Další nevýhodou může být, že data organizace či jednotlivce jsou v rukou třetí strany. Organizace často ani nemá ke svým datům přímý přístup, pouze prostřednictvím dané aplikace, což může zkomplikovat možnosti jejich dalšího zpracování. (Velte 2011)

Typy cloudových služeb lze dále rozdělit do tří skupin: Software jako služba (SaaS), Platforma jako služba (PaaS) a Infrastruktura jako služba (IaaS). Jednotlivé typy služeb budou dále popsány a jejich vzájemný vztah je zachycen na následujícím obrázku. (Microsoft 2018a)



Obrázek 4: Porovnání Saas, PaaS, IaaS
Zdroj: (Microsoft 2018a)

Software jako služba

Software jako služba (Software as a Service, zkráceně SaaS) spočívá ve využití internetového prohlížeče jako prostředí pro klientskou část aplikace. Software je dostupný jako služba prostřednictvím webových stránek. Serverová část je nasazena na cloudu, se kterým klientská část aplikace komunikuje. Software potom může být plně spravován organizací, která jej vyvíjí, bez ohledu na vůli uživatelů. Ta také nese veškeré náklady na jeho provoz. Poskytovatel služby potom samozřejmě také odpovídá za opravu. (Velte

2011) Většinou existuje několik úrovní služby, každá za jinak vysoký měsíční poplatek. Pro definování úrovně poskytovaných služeb se používá dokument nazývaný Service-Level Agreement (zkráceně SLA).

V této souvislosti se také mluví o architektuře softwaru orientované na služby (zkráceně SOA). Jedná se o přístup ke strukturování softwarového systému jako sady služeb. Tyto služby pak mohou být distribuované a poskytované různými poskytovateli. Použití SOA však nemusí znamenat, že software bude poskytován jako služba. Software může disponovat rozhraním, pomocí kterého služby spouští jiné aplikace. (Sommerville 2013)

Platforma jako služba

Platforma jako služba (Platform as a Service, zkráceně PaaS) je úplné prostředí pro vývoj a nasazení v cloudu. Umožňuje vyvíjet cokoliv od jednoduchých cloudových aplikací po podnikové informační systémy. Cloudová služba potom poskytuje technické prostředky, ke kterým se přistupuje pomocí zabezpečeného internetového připojení. (Velte 2011)

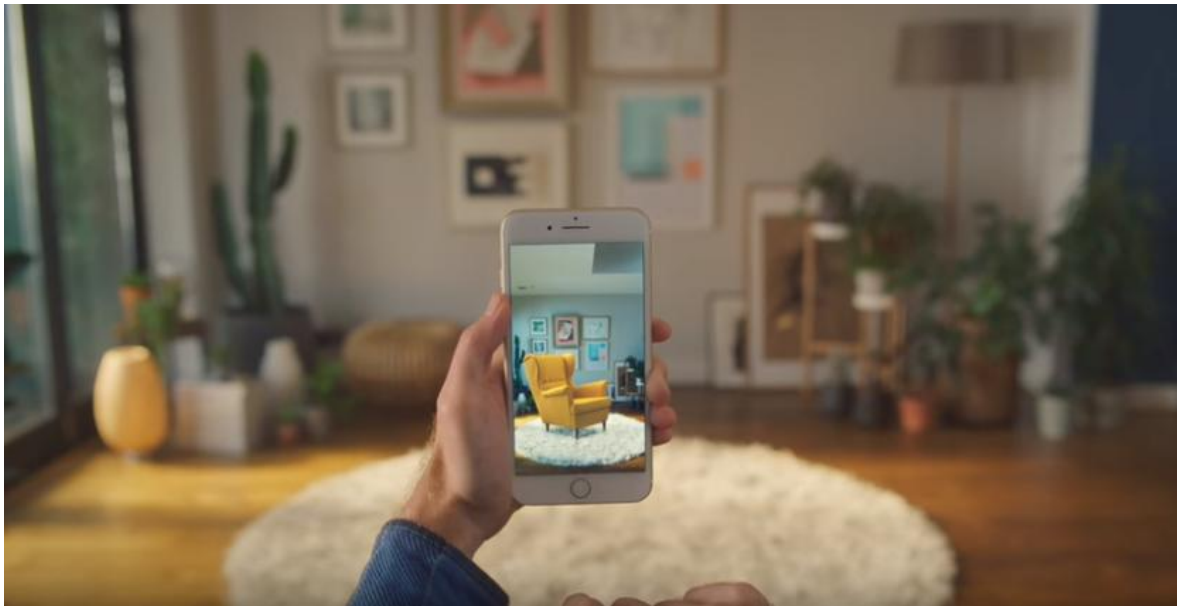
Infrastruktura jako služba

Infrastruktura jako služba (Infrastructure as a Service, zkráceně IaaS) je nejzákladnější formou cloudové služby. Jedná se o připravenou výpočetní infrastrukturu, která je poskytovaná přes internet. (Velte 2011)

2.1.6 Virtuální a rozšířená realita

Virtuální realita (zkráceně VR) je technologie, která umožňuje osobu přenést do umělého světa, který působí jako realita. Může se jednat o vytváření vizuálního, sluchového, hmatového či jiného zážitku, který budí subjektivní dojem skutečnosti. Různé ovladače a senzory navíc dokážou uživateli umožnit interakci s virtuálním prostředím či pohyb v něm. Virtuální realita se v současnosti používá v různých oborech, například v lékařství, armádě a především pro účely zábavy ve video hrách. Vývoj kompletní rozšířené reality je náročný a nákladný. Virtuální realita je obvykle realizovaná pomocí speciálních brýlí, v současnosti se rovněž experimentuje s možnostmi holografické projekce. (The Foundry 2018)

Rozšířená realita (Augmented Reality, zkráceně AR) spočívá v doplňování reálného obrazu o další informace nebo grafické objekty v reálném čase. Propracovanější systémy rozšířené reality umísťují virtuální objekty do reálného světa tak, aby to působilo přirozeně. Orientace kamery zařízení pro snímání reality se vyhodnocuje podle dat z čipu GPS (Global Positioning System), kompasu a akcelerometru. Dále tyto aplikace využívají rozpoznávání obrazu, aby dokázaly vyhodnotit, kam virtuální objekt vhodně umístit. Uplatnění rozšířené reality je především jako podpora prodeje, v herním průmyslu, ve vzdělávání a daších oborech. (The Foundry 2018) Na následujícím obrázku je možné vidět ukázku aplikace IKEA Place, která umožňuje vyzkoušet, jak budou jednotlivé produkty od této firmy vypadat v reálném prostředí. (IKEA 2018)



Obrázek 5: IKEA Place
Zdroj: IKEA 2018

Kombinovaná realita byla představena společností Microsoft jako Windows Mixed Reality. Má se jednat o jakýsi hybrid mezi virtuální a rozšířenou realitou. Cílem je nabídnout uživateli zážitek podobný skutečnosti, který bude moci sám ovládat. Nabízí nejen hraní her, ale také simulaci a prozkoumávání různých skutečných míst, sportovní simulátory a mnoho dalšího. Virtuální realita však v současnosti funguje obdobně a autorka této práce nenašla zásadní rozdíly mezi kombinovanou a virtuální realitou. Pro zpřístupnění kombinované reality si uživatel musí nasadit speciální headset, který oproti virtuální realitě obsahuje kamery, které snímají okolí a mohou buď přenášet reálné objekty do virtuální

reality, nebo virtuální prvky do obrazu skutečného světa. Dále je třeba headset zapojit do počítače, který podporuje Windows Mixed Reality a do rukou vzít speciální ovladače. (Voříšek 2017)

2.1.7 Využití biometrik

Biometriky je možné definovat jako „automatizované metody měření fyziologických a behaviorálních (týkající se chování) charakteristik osob za účelem zjištění nebo ověření identity těchto osob“. (Vach 2018)

V současnosti se stále více využívá biometrik k autentizaci uživatelů v různých aplikacích. Uživatelé si musí pamatovat obrovské množství hesel, a když dojde k jejich ztrátě či odcizení, přijdou o cenná data. Biometriky tomuto mohou zabránit. Uživatel je během pár vteřin identifikován podle některého z jedinečných fyziologických znaků. Tyto znaky přitom nelze napodobit jiným uživatelem. Nejčastěji se používá identifikace pomocí otisku prstu, který je přiložen na speciální senzor. (Yoshida 2018)

V současnosti existuje Web Authentication API, které má umožnit přihlašování ve webových aplikacích pomocí biometrik, zatím však není podporováno všemi prohlížeči. (W3C 2018)

2.1.8 Umělá inteligence

Umělá inteligence (Artificial Intelligence, zkráceně AI) se používá ve dvou souvislostech:

1. Umělá inteligence označuje schopnost stroje napodobit lidské přemýšlení a chování.
2. Umělá inteligence je obor informatiky, který se zabývá učením strojů umělé inteligenci.

Umělá inteligence se v oblasti ICT využívá například k rozpoznávání řeči, plánování a řešení problémů, předpovídání určité skutečnosti a dalším úkonům, které dříve musel provádět člověk. Předpokládá se, že se stroje s umělou inteligencí dokážou učit ze svých zkušeností. Umělá inteligence je v současnosti součástí sociálních sítí, prohlížečů

i vyhledávačů, hojně se využívají chatboti, kteří se učí na základě komunikace s uživateli, v mnoha aplikacích uživatel umělou inteligenci ani nevnímá. (Bernard 2018b)

Proces učení je velmi důležitý. Pokud jsou stroje učeny nesprávnými daty, mohou snadno začít dělat chyby. V posledních dvou letech se v médiích objevilo mnoho varování před umělou inteligencí. Největším problémem umělé inteligence je, že není možné přesně odvodit, proč se stroj rozhodl tak, jak se rozhodl, a chyby v naučeném rozhodovacím procesu lze odhalit až na základě špatných rozhodnutí. (Nývlt 2017)

2.1.9 Virtuální měny

Virtuální měny, označované také jako kryptoměny, slouží jako platidlo v digitálním světě. Jsou reprezentovány pomocí záznamů v databázi, které za normálních okolností nelze měnit. Záznamy jsou zapisovány do tzv. blockchainu. Ten je spravován pomocí decentralizované sítě ověřovatelů, kteří transakce ověřují. (Rosic 2017)

Virtuálních měn v současnosti existují stovky, mezi nejznámnější patří první kryptoměna Bitcoin, dále například Ethereum, Ripple nebo Litecoin. Princip blockchainu se začíná využívat i v jiných oblastech. (Rosic 2017)

2.2 Technické prostředky vývoje

V této části se diplomová práce zaměří především na technické prostředky, které jsou v současnosti využívány při vývoji aplikací.

2.2.1 Podpůrné softwarové nástroje

Při vývoji softwaru se často používají podpůrné softwarové nástroje Jejich cílem je automatizovat některé aktivity vývojového procesu. Podrobněji budou popsány pouze některé druhy těchto nástrojů, které přímo souvisí se samotným vývojem. V praxi se však podpůrný software používá téměř ve všech oblastech, využívají se modelovací nástroje, software pro plánování, analytické nástroje a další.

Framework

Framework je možné přeložit jako aplikační rámec, který usnadňuje vývoj tím, že předdefinuje určité procesy. Zahrnuje obvykle doporučené postupy a pravidla, podpůrné programy, sady funkčností, které se v aplikacích často opakují a každý vývojář by je jinak musel řešit sám, a k nim samozřejmě související dokumentaci. Framework se vždy váže k určitému programovacímu jazyku. (Shukla 2017)

API

Application Programming Interface, zkráceně API, je možné přeložit jako rozhraní pro programování aplikací. Jedná se o sadu funkčností, které je možné využít ve vyvíjené aplikaci. API obsahuje také postup, jak jednotlivé funkčnosti používat. Většinou jsou určeny pro konkrétní programovací jazyk, ale nemusí tomu tak vždy být. V případě, že se nejedná o obecné funkčnosti, ale o přístup k nějaké konkrétní službě (například různé databáze, registry atd.), je vhodné, aby API bylo univerzální a jazykově nezávislé. (Shukla 2017)

Representational State Transfer (zkráceně REST), označuje architekturu pro tvorbu API nezávislého na platformě. Jednotlivé příkazy, které jsou součástí API, je možné spouštět pomocí internetových přenosových protokolů, nejčastěji se používá HTTP/HTTPS. Takové API bývá označováno jako REST API, případně RESTful API. (Stackify 2017)

SDK

Software Development Kit, zkráceně SDK, je sada vývojových nástrojů určená pro programování aplikací pro určitý programovací jazyk či platformu. Umožňuje kód kompilovat či exportovat do různých formátů, obsahuje dokumentaci, doporučené postupy, API atd. Příkladem může být Java Development Kit (zkráceně JDK), který obsahuje základní nástroje pro vývoj v jazyce Java. (Shukla 2017)

Vývojové prostředí

Vývojové prostředí (Interactive Development Environment, zkráceně IDE) je software, ve kterém vývojáři píší zdrojový kód. Vývojové prostředí má za cíl tento proces co nejvíce usnadnit. Nabízí například tyto nástroje:

- vývojářské textové editory, které kontrolují syntaxi a vizuálně odlišují jednotlivé prvky programovacího jazyka,
- formátování či rychlý refaktoring kódu, například při přejmenování nebo přesunu souborů,
- zobrazení struktury aplikace a navigace uvnitř komponent, což zahrnuje například otevření zdrojového kódu funkce z místa, kde je použita,
- nástroje pro použití verzovacích systémů,
- zobrazování změn v kódu a možnost zobrazení historie,
- nástroje pro ladění programu,
- nástroje pro automatické testování jednotlivých částí kódu pomocí jednotkových testů,
- a další nástroje, které vývoj urychlí, zefektivní a usnadní.

Příkladem vývojového prostředí je například NetBeans IDE, které je zdarma a podporuje hned několik programovacích jazyků i operačních systémů. Dalším příkladem může být open source vývojové prostředí Eclipse. Mnoho vývojových prostředí pro různé programovací jazyky nabízí společnost JetBrains, jejich produkty jsou ale placené. (Guru99 2018)

Verzovací systémy

Verzovací systémy (Version Control System, zkráceně VCS) jsou nástroje, které umožňují zdrojový kód aplikace sdílet mezi jednotlivými členy týmu, archivovat a porovnávat vůči starším verzím. Jejich služby se využívají pomocí příkazového řádku, ale vývojová prostředí obvykle disponují možností, jak verze spravovat za pomoci grafického uživatelského rozhraní. Příkladem může být Subversion, založený na architektuře klient-server, nebo distribuované systémy Git a Mercurial. (Ducho 2014)

2.2.2 Přístupy k programování

Způsob, jakým se psal zdrojový kód, a jak byly jeho jednotlivé části strukturovány, se postupně vyvíjel.

Strojový kód

Nejprve se kód psal přímo ve strojovém jazyce. To bylo značně nepraktické, protože kód nebyl čitelný lidským okem, nebylo možné pojmenovávat proměnné a přinášelo to mnoho dalších nevýhod. (Žůrek, 2018)

Nestrukturované paradigma

Tento postup již umožňoval vytvářet komplexnější programy, ale jeho využití nebylo příliš praktické. Počítač zdrojový kód vykonával postupně odshora dolů, případně byl možný skok na požadované místo v kódu na základě podmínky. Takový kód byl nepřehledný, i když již částečně srozumitelný lidskému oku. (Žůrek 2018)

Strukturované paradigma

Tento přístup již akceptoval, že vyvoj komplexních a rozsáhlých aplikací vyžaduje, aby byl kód nějakým způsobem strukturován a aby stále se opakující postupy nebylo nutné definovat pokaždé znovu. Strukturované paradigma potom umožňovalo členění kódu do jednotlivých funkcí, které bylo možné volat z jiných funkcí pomocí zadání názvu funkce a parametrů. Později byla přidána možnost funkce seskupovat do modulů, což bývá označováno jako modulární programování. (Žůrek 2018)

Objektově orientovaný přístup

V současnosti je nejčastěji využíván objektově orientovaný přístup k programování (Object-oriented programming, zkráceně OOP). Snaží se kód strukturovat s cílem maximalizovat jeho udržovatelnost a znovupoužitelnost. Objektově orientovaný přístup představuje poněkud jiný způsob strukturování kódu. Aplikace je složena z jednotlivých komponent, nazývaných třídy, které mohou vzájemně komunikovat. Třída je potom zobecněním určitého objektu reálného světa. V kódu se může objevit například třída Order (objednávka). Jednotlivé výskyty třídy jsou označovány jako její instance. Objednávka č. X je potom instancí třídy Order. Každá třída má definované atributy a metody. Jednotlivé instance se liší hodnotami atributů, přičemž metody jsou pro ně společné. Jejich návratová hodnota se ale opět může lišit. (Žůrek 2018)

Objektově orientovaný přístup je založen na čtyřech základních principech:

- *Zapouzdření* znamená, že je možné některé atributy a metody třídy schovat tak, aby je bylo možné používat jen zevnitř. Navenek potom třída působí jako černá skříňka, ke které se zvenčí přistupuje pouze pomocí rozhraní, které zahrnuje pouze *veřejné* (public) metody dané třídy. Uvnitř jsou schované *privátní* (private) metody a atributy, ty může použít jen daná třída, a *protected* metody a atributy, které může použít daná třída a její potomci.
- *Dědičnost* zahrnuje odvozování tříd podle jiných, obecnějších, což vede k lépe udržovatelnému kódu. Obecnější třída je označována jako předek a třída, která od ní dědí, jako potomek. Je možné dědit jak atributy, tak metody dané třídy.
- *Polymorfismus* umožňuje definovat mnohotvárné metody, které mají jeden název, ale mohou se volat s různými typy parametrů a chovají se různě podle toho, k jaké třídě patří a jak jsou použity. Zahrnuje také možnosti úpravy metod předka pro potřeby potomka.

Ne každý však považuje OOP za ideální. V současnosti se diskutuje o tom, zda je vhodné využívat objektový přístup nebo to jen zbytečně přidělává práci, zvyšuje množství kódu a souvisejících dat a zhoršuje udržitelnost. Alternativou OOP je procedurální programování. Na objektově orientovaném přístupu jsou však založeny nejpoužívanější programovací jazyky, takže prozatím budoucnost určitě má. (Rewallen 2005)

2.2.3 Softwarové modelování

Softwarové modelování je jednou z částí návrhu softwaru. Slouží k tvorbě abstraktního modelu budoucího produktu. Modely obvykle využívají grafické prvky, které mají za cíl zápis zestručnit a zpřehlednit. Modelování podporuje celý proces vývoje. Na úplném začátku pomáhá odvodit požadavky na systém, ve fázi návrhu slouží k popsání technik, které jsou v další fázi implementovány, po implementaci pomáhá s tvorbou dokumentace a ke kontrole, zda je aplikace v souladu s návrhem. Modelování lze využít jak při vývoji nové aplikace, tak při rozvoji již existujícího softwaru. (Sommerville 2013)

Ke světově uznávaným standardům v oblasti softwarového modelování patří modelovací jazyk Unified Modeling Language, zkráceně UML. Jazyk UML definuje standardizační nezisková organizace Object Management Group, zkráceně OMG. Nabízí diagramy

použitelné pro různé fáze vývoje a opakovaně ověřované praxí. Existují dokonce nástroje, které z některých diagramů, například z diagramu tříd, mohou přímo generovat zdrojový kód a vývojáři potom stačí vyplnit definice jednotlivých metod. (OMG 2017)

Součástí UML je sada třinácti diagramů, které umožňují modelovat softwarové systémy. Tyto diagramy jsou rozděleny do tří skupin: (OMG 2017)

- Diagramy struktury
    - Diagram tříd (*Class Diagram*)
    - Objektový diagram (*Object Diagram*)
    - Komponentový diagram (*Component Diagram*)
    - Diagram složených struktur (*Composite Structure Diagram*)
    - Balíčkový diagram (*Package Diagram*)
    - Diagram nasazení (*Deployment Diagram*)
- Diagramy chování
    - Diagram případů užití (*Use Case Diagram*)
    - Diagram aktivit (*Activity Diagram*)
    - Stavový diagram (*State Machine Diagram*)
- Diagramy interakce
    - Sekvenční diagram (*Sequence Diagram*)
    - Komunikační diagram (*Communication Diagram*)
    - Diagram časování (*Timing Diagram*)
    - Diagram interakcí (*Interaction Overview Diagram*)

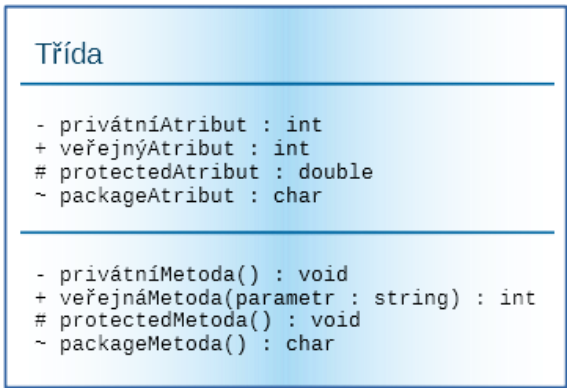
V následující části diplomové práce bude popsán zástupce každé z těchto skupin: objektový diagram, diagram případů užití a sekvenční diagram.

Diagram tříd

Jedná se o diagram, který zachycuje strukturu aplikace – její třídy, atributy a metody včetně vstupních parametrů a návratových hodnot. Návrh struktury by měl být co nejpřesnější, aby podle něj bylo možné aplikaci implementovat. Názvové konvence se odvíjejí podle programovacího jazyka, který bude použit pro implementaci aplikace.

Na následujícím obrázku je zachycena reprezentace třídy. V první části je uveden název třídy. V druhé části jsou uvedeny atributy a ve třetí části metody dané třídy. U atributů se uvádí název, datový typ a také modifikátor přístupu. Název atributu a datový typ je oddělen dvojtečkou. U metod se uvádí rovněž název a modifikátor přístupu, ale navíc i parametry a jejich datové typy a typ návratové hodnoty. Mezi názvem parametru a jeho datovým typem se píše dvojtečka, která se rovněž píše mezi metodou a typem návratové hodnoty. Název atributu a datový typ je oddělen dvojtečkou. Modifikátor přístupu je definován prvním znakem u atributu či metody a může mít jednu ze čtyř hodnot: (Čápka 2018)

- Mínus (-) značí privátní (private) atribut.
- Plus (+) je označením pro veřejný (public) atribut.
- Křížek (#) značí protected atribut.
- Tilda (~) označuje atribut viditelný v rámci balíku (package).



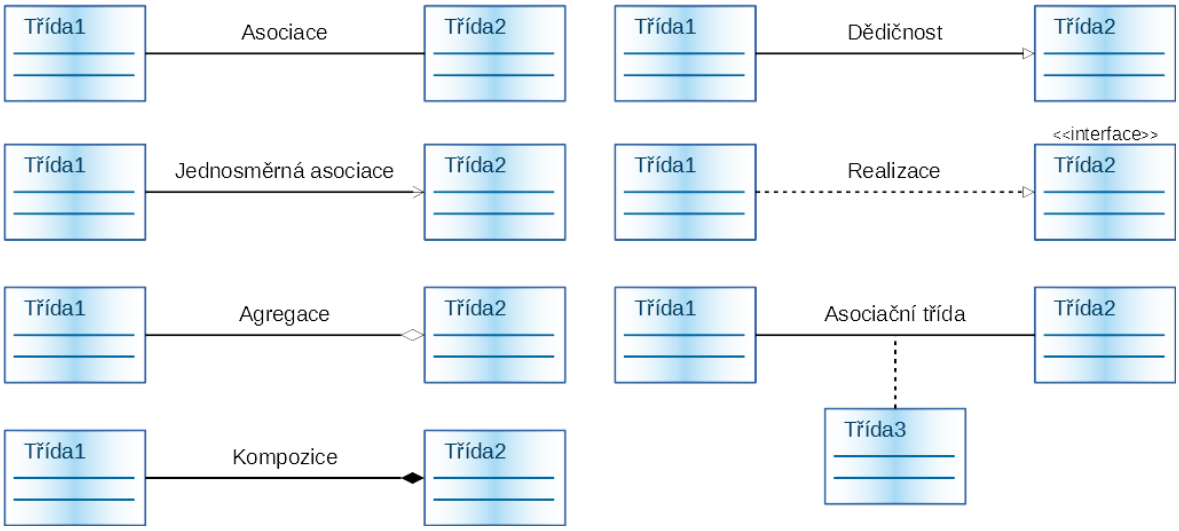
Obrázek 6: Třída
Zdroj: Čápka 2018

Vztahy mezi jednotlivými třídami jsou udávány různými typy vazeb. (Čápka 2018)

- Prvním typem vazby je *asociace*. Reprezentuje vztah dvou entit, které mohou existovat nezávisle na sobě. Grafickým znázorněním je plná čára spojující obě třídy. Asociace může propojit i více tříd.
    - Pokud je vazba jednosměrná, je čára reprezentující vazbu zakončená šipkou směřující ze třídy, která si uchovává odkaz. Na následujícím obrázku tedy Třída1 uchovává odkaz na Třídu2.
- Druhým typem vazby je *agregace*. Jedná se o vztah, kdy je jedna ze tříd součástí té druhé. Grafickým znázorněním je plná čára zakončená prázdným kosočtvercem

u třídy, která reprezentuje celek. Na následujícím obrázku je Třída1 součástí celku, kterým je Třída2.

- Dalším typem vazby je *kompozice*. Je podobná agregaci, ale reprezentuje silnější vztah, část nemůže bez celku existovat. Na následujícím obrázku je stejně jako v případě agregace Třída1 součástí celku, kterým je Třída2.
- Čtvrtým typem vztahu je *generalizace*, která reprezentuje dědičnost. Graficky je znázorněna pomocí plné čáry, která je na straně předka zakončena prázdným trojúhleníkem. V případě zachyceném na následujícím obrázku dědí Třída1 vlastnosti předka Třída2.
- Pátý typ vztahu, *realizace*, se používá pro popis vazby mezi třídou a interface (viz kapitola 2.2.1), na který se třída změní pomocí tzv. *stereotypu* (funkce jazyka UML, která slouží k pozměnění významu prvků diagramu na nový význam, který je uveden ve dvojitých špičatých závorkách). Na následujícím obrázku Třída1 používá interface Třída2.
- Dalším, trochu specifickým typem vztahu je *asociační třída*, která pomáhá popsat vazbu mezi dvěma třídami. Příkladem může být vztah mezi třídou „běžec“ a třídou „závod“, který pojí asociační třída „registrace“. Na následujícím obrázku Třída3 obsahuje pomocné atributy pro popsání vazby mezi Třídou1 a Třídou2.



Obrázek 7: Vazby mezi třídami
Zdroj: Čápka, 2018

U asociace, agregace a kompozice se navíc uvádí také *multiplicita*, která určuje, kolik instancí jedné třídy může být ve vazbě s instancí třídy druhé. Multiplicita se uvádí

k jednotlivým třídám ve vztahu, a pokud není uvedena, potom se předpokládá, že ve vztahu je právě jedna instance dané třídy. Multiplicitu je možné uvádět jedním z následujících způsobů: (Čápka 2018)

- Číselně či pomocí znaku hvězdičky (*), alternativně také pomocí písmene N, což znamená libovolný počet (včetně nuly).
- Intervalem, který je reprezentován dvěma hodnotami oddělenými dvěma tečkami (X..Y). Interval může též obsahovat symbol hvězdičky jako horní hranici.
- Výčtem několika hodnot z předchozích dvou bodů, oddělených čárkami (například 1, 3..5, 7..*).

Diagram případů užití

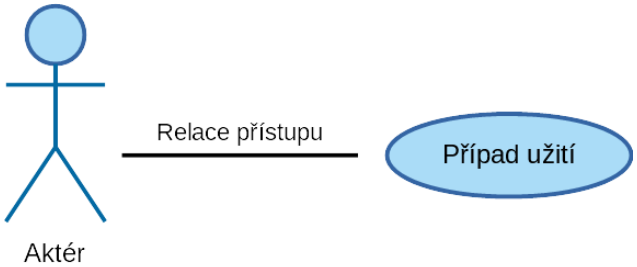
Diagram případů užití popisuje funkcionalitu systému. Je založen na aktérech, kteří mohou spouštět případy užití, a na vazbách mezi nimi.

Případ užití (Use Case) je sada akcí, které vedou k dosažení určitého cíle. Případ užití je graficky znázorněn pomocí elipsy, která obsahuje název funkcionality. V diagramu nejsou zobrazeny jednotlivé akce, které jsou během případu užití vykonávány, případ užití je zachycen pouze jako celek. Případy užití vyplývají z požadavků definovaných před zahájením modelování. V tomto diagramu by neměl být specifikován způsob, jakým jsou případy užití prováděny, ale jejich účel. Vzhledem k tomu, že diagram zachycuje pouze jednotlivé funkcionality a aktéry, kteří je mohou spustit, je vhodné ho doplnit o specifikaci případu užití. Ta obsahuje kromě názvu případu užití šest základních částí, které mají funkčnost podrobněji specifikovat. Společně s grafickým návrhem potom podává ucelený obrázek o funkčnostech softwaru. Jedná se o tyto části: (Čápka 2018)

- *Krátký popis* funkčnosti má za cíl podrobněji popsat funkčnost, ale tak, aby tomu porozuměl uživatel.
- *Aktéři* jsou zobrazeni jako seznam rolí, které mohou funkčnost spustit.
- *Podmínky pro spuštění* popisují stav, který musí nastat, aby bylo možné funkčnost spustit.
- *Základní scénář*, též hlavní scénář nebo Happy Day scénář, popisuje sekvenci akcí, které jsou součástí případu užití, v případě úspěšného průchodu celým algoritmem.

- *Alternativní scénáře* potom popisují, jak se má systém zachovat, pokud dojde během některé z akcí k chybě nebo k nesplnění podmínek pro průchod základním scénářem. Mohou popisovat i alternativní akce, které jsou podmíněny specifickými podmínkami. Alternativní scénáře se potom odvíjejí podle očekávaných faktorů, které mohou vést k odchýlení od základního scénáře. Mohou jimi být například chybná vstupní data, nesplněné vstupní podmínky, nenalezení záznamu v databázi atd.
- *Následné podmínky* definují stav po dokončení případu užití.

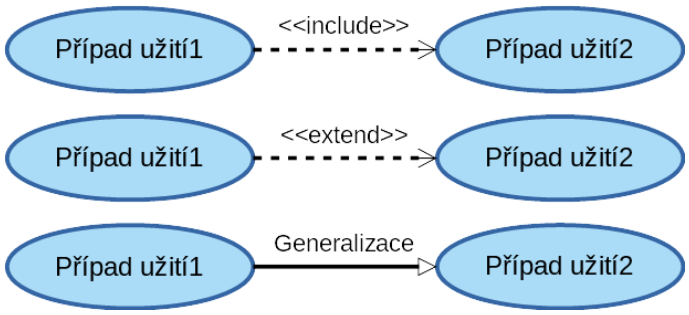
Aktér (Actor) je role, která komunikuje s aplikací prostřednictvím případů užití. Může to být nejen uživatel, ale také například jiný systém. Aktivní aktéři, kteří komunikaci iniciují, jsou zobrazeni nalevo od případu užití, pasivní aktéři, se kterými zahajuje komunikaci aplikace, se zakreslují napravo.



Obrázek 8: Aktér, relace a případ užití
Zdroj: Čápka 2018

Relace reprezentují vztahy mezi případy užití a aktéry. Používány jsou čtyři základní typy vazeb: (Čápka 2018)

- Relace *přístupu*, která spojuje aktéry s případy užití, které mohou spouštět. Grafickým znázorněním je plná čára, může být zakončena šipkou.
- Relace <<*include*>> slouží pro zahrnutí případu užití do jiného případu užití. Značí se přerušovanou čárou zakončenou šipkou, která směřuje k zahrnutém případu užití.
- Relace <<*extend*>> popisuje možnost rozšíření případu užití jiným případem užití. Značí se přerušovanou čárou zakončenou šipkou směřující k rozšiřujícímu případu užití.
- Relace *specializace/generalizace* vyjadřuje dědičnost mezi objekty (mezi aktéry i mezi případy užití). Je znázorněna plnou čárou zakončenou prázdným trojúhelníkem, která směřuje k objektu, od kterého se vlastnosti dědí.



Obrázek 9: Relace mezi případy užití
Zdroj: Čápka 2018

Sekvenční diagram

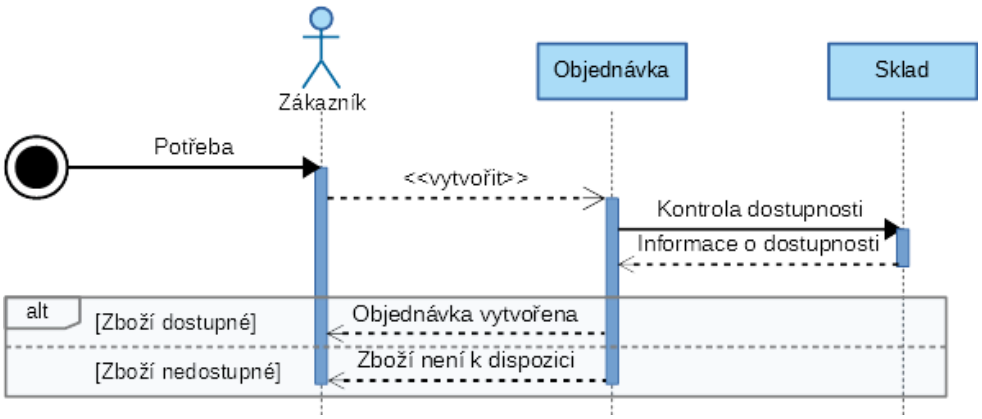
Sekvenční diagram zachycuje postupnou interakci mezi skupinou objektů v čase. Objekty mohou být jednotlivé třídy či aktéři a v sekvenčním diagramu jsou označováni jako klasifikátory. Sekvenční diagram obvykle zachycuje interakce související s jedním případem užití.

Jednotlivé klasifikátory jsou uvedeny vedle sebe v horní části diagramu. Svisle od nich vede přerušovaná čára. Část z ní je čára života (Lifeline) daného klasifikátoru, která je zobrazena pomocí obdélníku na přerušované čáře a reprezentuje čas, po který se instance objektu účastní výpočtu. Interakce mezi objekty jsou seřazeny postupně shora dolů a jsou znázorněny pomocí šipek. Pokud se jednotlivé interakce liší dle podmínek, je daný úsek případu užití ohraničen rámečkem s popisem „alt“. Ten je dále rozdělen na jednotlivé řádky oddělené přerušovanou čarou, na jejichž začátku je v hranatých závorkách definovaná podmínka pro vykonání daného řádku. Typ jednotlivých interací se liší podle stylu spojovací čáry (Microsoft 2018b):

- Synchronní zpráva je zobrazena pomocí plné čáry zakončené šipkou.
- Asynchronní zpráva je zobrazena pomocí plné čáry zakončené plným trojúhelníkem.
- Zpráva s odpovědí je reprezentována přerušovanou čarou zakončenou šipkou či plným trojúhelníkem.
- Vytvoření objektu je reprezentováno přerušovanou čarou zakončenou šipkou.
- Odstranění objektu je zobrazeno pomocí přerušované čáry zakončené šipkou, za kterou následuje symbol kříže ve tvaru x.
- Komunikace klasifikátoru se sebou samým je znázorněna pomocí plné čáry zakončení šipkou, která směřuje z daného objektu, otáčí se a vrací se zpět k němu.

- Nalezená zpráva, tedy zpráva odeslaná neznámým odesilatelem, je graficky zobrazena pomocí plné řáry zakončené šipkou směřující od počátečního bodu, který je reprezentován kruhem.
- Ztracená zpráva, tedy zpráva s neznámým příjemcem, je rezeprezntována plnou čarou, na jejímž konci je šipka a která směřuje ke koncovému bodu, který je graficky znázorněn stejně jako počáteční bod pomocí kruhu.

Na následujícím obrázku je možné vidět ukázku sekvenčního diagramu.



Obrázek 10: Sekvenční diagram
Zdroj: Microsoft 2018b

2.2.4 Volba platformy

Co to vlastně softwarová platforma je? Jedná se o dost abstraktní pojem a není stanovena jasná hranice mezi tím, co je software a co už je softwarová platforma, software může být i platformou. Platforma je vlastně pracovní prostředí a to jak po stránce hardwarové, tak po stránce softwarové. (Bridgwater 2015)

Vývojovou platformou bývají označována vývojová prostředí či frameworky určené pro vývoj software včetně programovacího jazyka, knihoven či jiných komponent. Příkladem takové platformy může být například Java nebo .NET od Microsoftu.

Platformou může být také operační systém či konkrétní zařízení (například iPhone). Některé platformy jsou vázány ke konkrétnímu hardware a bez něj fungovat nemohou, jiné platformy se snaží o maximální nezávislost na hardware.

Způsob vývoje aplikace by měl být přizpůsoben všem cílovým platformám, pro které je aplikace určena. Samozřejmě je možné vyvíjet nejprve pro jedno konkrétní zařízení a poté software rozšiřovat, ale většinou to přinese dodatečné náklady.

Pro potřeby této kapitoly budou používány následující pojmy:

- *Zařízením* se myslí jakýkoliv počítač, který je schopný samostatně fungovat, ať už se jedná o desktopový počítač, tablet či chytré hodinky.
- Pojem *platforma* pro usnadnění bude označovat právě vývojové platformy, které byly vytvořeny jako nástroj pro budování nového software nebo alespoň pro jeho provoz či distribuci, a ovlivňují možnosti volby technologií, které mohou být při vývoji použity. Platformou může být i internetový prohlížeč či operační systém.

Volba platformy je důležitá především s ohledem na koncové uživatele. Je nepravděpodobné, že někdo zakoupí zařízení jen kvůli tomu, aby mohl používat určitou aplikaci. Ideální je, aby vyvíjená aplikace podporovala co nejvíce různých zařízení, operačních systémů, ale samozřejmě čím více platforem bude podporovat, tím náročnější a nákladnější bude vývoj a testování aplikace. (Bridgwater 2015)

2.2.5 Druhy aplikací

V souvislosti s volbou platformy je třeba rozdělit aplikace do několika druhů podle toho, kde jsou uložena programová data aplikace a jak je k nim přistupováno. Pro potřeby této práce jsou definovány tyto základní druhy aplikací:

- desktopové (nativní) aplikace
- webové aplikace
- progresivní webové aplikace
- hybridní aplikace

Desktopové (nativní) aplikace

Jedná se o nejstarší typ aplikací, které existovaly již před rozšířením internetu. I v dnešní době však najdou své využití. Jedná se o takovou aplikaci, která je napsaná pro konkrétní platformu a před spuštěním je třeba ji nainstalovat na cílový počítač. (iQuest 2017)

První nativní aplikace připojení k internetu nevyužívaly vůbec. S příchodem internetu se začalo využívat při vývoji desktopových aplikací architektury klient-server (více v kapitole 2.6.1). Princip této softwarové architektury spočívá v tom, že si cílový uživatel nainstaluje klientskou část do svého zařízení a aplikace se při spuštění připojí na server. Z ní se vyvinuly další druhy softwarových architektur. Při rozdělení aplikace na klientskou a serverovou část má provozovatel aplikace kontrolu nad serverovou částí a do jisté míry může ovlivnit i část klientskou (například nabídnout instalaci nové verze), ale v míře omezené tím, jaké funkce byly implementovány při vývoji klientské části.

Hlavní výhodou nativní aplikace je, že může dosáhnout maximální interakce s uživatelem především s ohledem na cílové zařízení (možnost zobrazovat upozornění na mobilním telefonu, umístění zástupce na ploše, možnost komunikovat s hardwarem daného zařízení). Výkon takové aplikace není omezen prohlížečem a u offline aplikací ani internetovým připojením, ale pouze výkonem zařízení, na kterém je aplikace nainstalována. (Sommerville 2013)

Nativní aplikace jsou vytvořené pro jednu konkrétní platformu a instalují se přímo na cílové zařízení. Rozšíření aplikace na jiné platformy znamená napsat aplikaci znovu, pomocí jiných postupů a programovacích jazyků. Vývoj nativní aplikace pro různé platformy je výrazně nákladnější než u jiných druhů aplikací a je vhodnější ho zvolit, pokud se zaměřujeme jen na jednu platformu. Největší nevýhodou je, že tvůrce aplikace nemá nad nainstalovanou verzí kontrolu a pokud by bylo potřeba něco pozměnit či opravit, cílový uživatel by musel nainstalovat opravný balíček či novou verzi aplikace. Především před rozšířením internetu byla i samotná distribuce opravného balíčku nákladnou záležitostí, přinejmenším v ceně datového média a dopravy. Nativní aplikace bez připojení k internetu neměly žádnou kontrolu nad uživatelskými daty a nemohly zaručit jejich bezpečnost. (iQuest 2017)

Webové aplikace

Jedná se o aplikaci, která je přístupná prostřednictvím internetu, obvykle jako webová stránka zobrazovaná internetovým prohlížečem. (MobiLoud 2018)

Při vývoji webových aplikací je nutné zamyslet se nad tím, jak se stránka zobrazí na monitorech různých rozlišení a především na mobilních zařízeních. V této souvislosti se hovoří buď o mobilní verzi webové stránky, většinou s možností přepnout na desktopovou verzi, a také o responzivním designu, který spočívá v tom, že se uživatelské rozhraní aplikace přizpůsobí danému zařízení, aby bylo její ovládání co nejpřívětivější pro cílové uživatele. (Sharkie 2015)

V souvislosti s webovými aplikacemi se užívají pojmy frontend a backend. Jedná se o dvě oddělené části webové aplikace. Části viditelné pro uživatele se říká frontend a zajišťuje veškerou interakci s uživatelem. Části pro zpracování dat se říká backend a má na starosti mimo jiné správu datového úložiště. Každá z těchto částí může využívat rozdílné technologie, je však třeba myslet na to, aby byly vzájemně kompatibilní, protože backend i frontend mezi sebou nutně musí komunikovat a předávat si data.

Oproti nativním aplikacím mají mnoho výhod, například (iQuest 2017):

- cílový uživatel nemusí nic instalovat,
- aplikace včetně uživatelských dat je dostupná odkudkoliv,
- provozovatel aplikace má kontrolu nad celou aplikací,
- verze pro různé platformy jsou vyvíjeny současně,
- opravy mohou být nasazeny nezávisle na vůli uživatelů,
- výpočetní operace probíhají na serveru a nezpomalují cílové zařízení,
- aplikace nezabírá místo v paměti cílového zařízení,…

Nevýhodou je, že nejsou dostupné bez internetového připojení. Problémem webových aplikací je i obtížná komunikace s hardware cílového počítače.

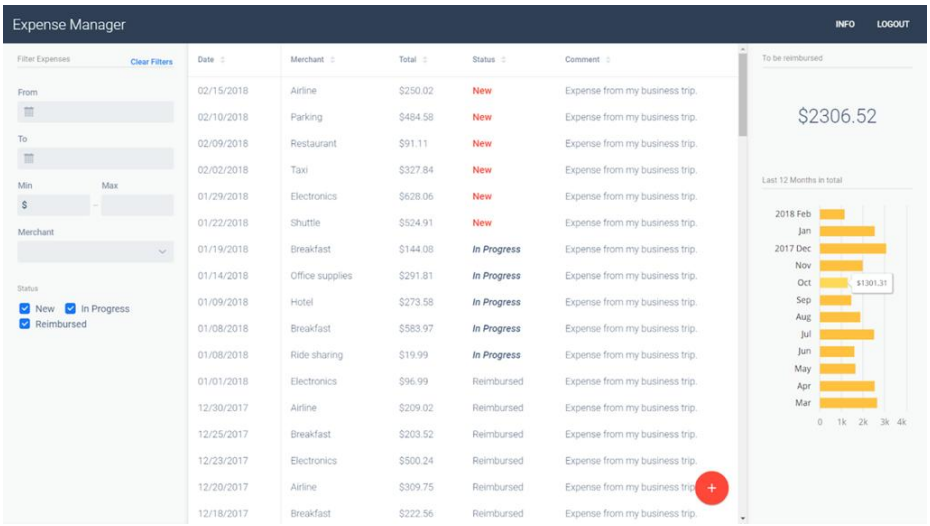
Progresivní webové aplikace

Jedná se o moderní typ webových aplikací představený společností Google v roce 2015. Aplikace jsou dostupné prostřednictvím webového prohlížeče, ale snaží se cílovému uživateli nabídnout stejný uživatelský zážitek jako u nativních aplikací. (Google 2018)

Toto řešení se snaží odstranit většinu nevýhod nativních a webových aplikací a ponechat z nich to nejlepší. Po stažení malého množství dat při prvním spuštění je aplikace dostupná i offline. Uživateli automaticky nabídne přidání zástupce na domovskou obrazovku. Aplikaci není nutné instalovat a je možné se na ni odkázat z jiných webových stránek. Svůj vzhled aplikace přizpůsobí cílovému zařízení, jako tomu je u responzivních webových aplikací. (Google 2018)

Nevýhodou je, že se pořád jedná o webovou aplikaci, která nemá přístup k některým funkcím zařízení (sms, kontakty) či k hardwaru, výkon je omezen prohlížečem a internetovým připojením.

Na následujících obrázcích je porovnáno zobrazení dema progresivní webové aplikace Expense Manager. Nejprve na desktopu (se zobrazením prohlížeče na celou obrazovku), poté na mobilním telefonu.



Obrázek 11: Expense manager desktopové zobrazení
Zdroj: Vaadin 2018

Obrázek 12: Expense manager mobilní zobrazení
Zdroj: Vaadin 2018

Hybridní aplikace

Hybridní aplikace jsou opět určitou kombinací nativních a webových aplikací. Vyžadují instalaci do cílového zařízení a působí jako nativní aplikace. Uživatel si do svého zařízení ale nainstaluje v podstatě jen zjednodušenou verzi webového prohlížeče (tzv. Webview), který zobrazuje webový obsah. (MobiLoud 2018)

Tento druh aplikace přebírá výhody webových aplikací. Působí jako nativní aplikace, což může přinášet také některé výhody, například pro uživatele dostupnost na speciálních obchodech s aplikacemi pro určité platformy. (iQuest 2017)

Nevýhodou může pro některé uživatele být nutnost instalace. Tím, že se jedná pouze o webový obsah, je stále omezena možnost komunikace s hardwarem a využití některých funkcí zařízení. Aplikace nepoužívá nativní ovládací prvky dané platformy (tlačítka, výběrové prvky apod.), ale vlastní HTML prvky.

Aplikace pro jiné platformy

Do této skupiny spadají všechny ostatní aplikace. Ať už se jedná o aplikace pro zařízení, která nemají operační systém a vyžadují využití speciálního programovacího jazyku, nebo o cloudové služby, které umožňují vývoj vlastní aplikace, která je pak provozovaná prostřednictvím dané služby. Příkladem může být aplikace pro Facebook, která se vyvíjí pomocí SDK nástrojů poskytnutých přímo od Facebooku.

2.2.6 Programovací jazyky

Algoritmy popisující fungování systému se vyjadřují pomocí nějakého programovacího jazyka. Ten slouží k zápisu programu v takové formě, které počítač rozumí. Programovacích jazyků existuje velké množství, ale jen některé z nich se dodnes těší popularitě mezi vývojáři. Stejně tak existuje několik druhů programovacích jazyků, které se liší dle způsobu zápisu zdrojového kódu a jeho struktury. (Gála 2009)

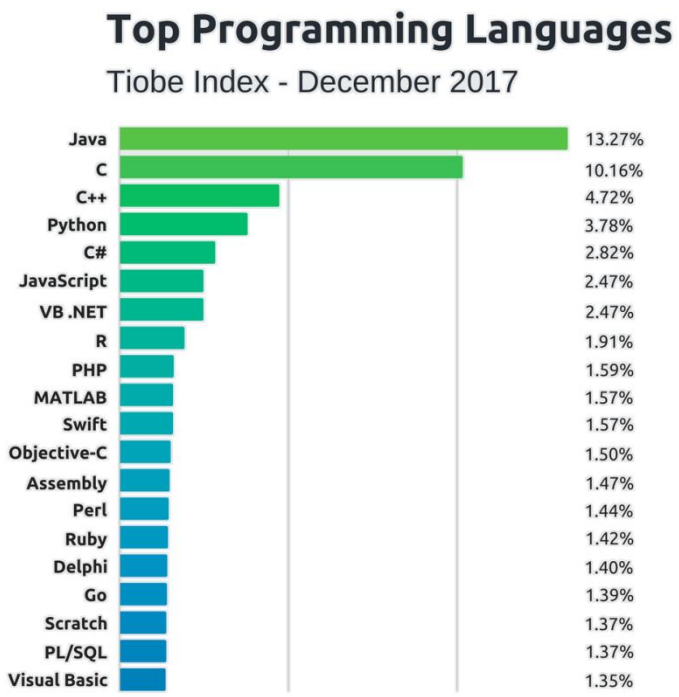
Programovací jazyky lze dělit na deklarativní a imperativní. (McGinnis 2016)

- *Deklarativní* programovací jazyky jsou definovány pomocí sekvence příkazů, které říkají, co se má stát, ale nespecifikují, jakým způsobem. Deklarativní jazyky mají několik podskupin, mezi ně patří programovací jazyky logické a funkcionální. Deklarativními programovacími jazyky se tato práce zabývat nebude.
- *Imperativní* programovací jazyky jsou definovány pomocí algoritmů, které popisují posloupnosti příkazů. Podle přístupu ke strukturování kódu je lze rozdělit na procedurální a objektově orientované. Většina procedurálních jazyků, které se využívají v současnosti, podporuje také objektově orientovaný přístup.

Programovací jazyky se dále dělí na jazyky psané strojovým kódem, kompilované a interpretované jazyky. U *strojového kódu* není nutné překládat kód pro použití počítačem, v ostatních případech vývojář píše kód v poměrně srozumitelném jazyce, který bude následně převeden na strojový kód. Překlad *kompilovaných* jazyků probíhá po jeho napsání a následně je spouštěn zkompilovaný kód. U *interpretovaných* jazyků probíhá překlad opakovaně při každém spuštění programu. (Žůrek 2018)

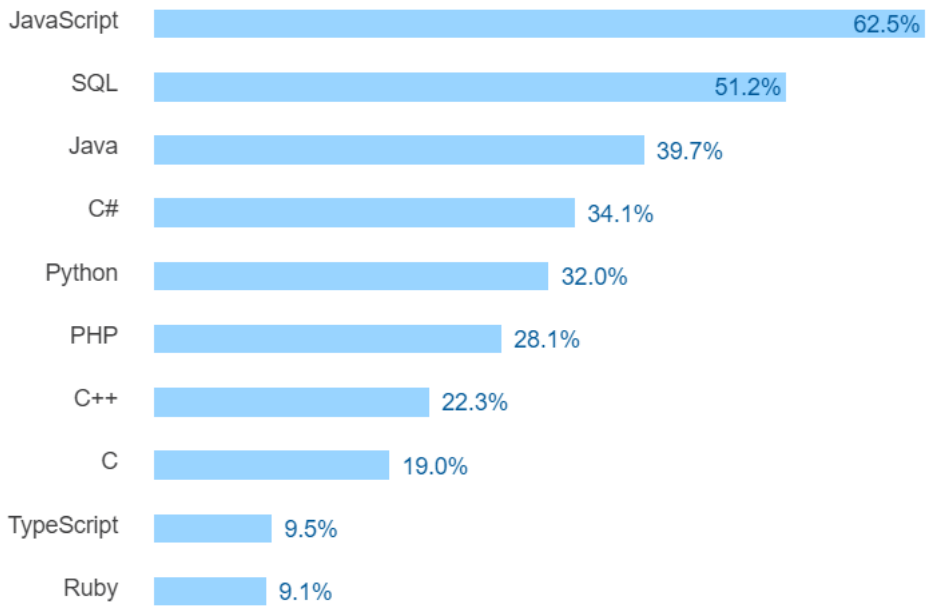
Tato práce se bude dále zabývat několika programovacími jazyky, které jsou v současnosti nejčastěji využívány.

Na obrázku níže je možné vidět žebříček nejpoužívanějších programovacích jazyků z prosince 2017, který sestavuje společnost Tiobe zaměřená na měření kvality softwarových produktů. Některé z nejoblíbenějších programovacích jazyků budou dále popsány. (Putano 2017)



Obrázek 13: Nejpoužívanější programovací jazyky Tiobe
Zdroj: Putano 2017

Další zajímavý průzkum proběhl na stránce StackOverflow, což je v podstatě sociální síť pro ICT odborníky. V tomto průkumu jednoznačně vyhrál JavaScript a výsledky je možné vidět na obrázku níže.



Obrázek 14: Nejpoužívanější programovací jazyky StackOverflow
Zdroj: StackOverflow 2017

Java

Java je jedním z nejoblíbenějších programovacích jazyků již od jejího vzniku. Jedná se o objektově orientovaný kompilovaný jazyk vyvinutý firmou Sun Microsystems v roce 1995. Hlavní výhoda tohoto jazyka spočívá ve snadné přenositelnosti aplikací vyvinutých v tomto jazyce – nejsou spuštěny přímo v operačním systému, ale prostřednictvím platformy Java, kterou je třeba mít v počítači nainstalovanou. (Pawlan 1999) Jedním z nejvyužívanějších frameworků pro jazyk Java je JUnit, který má za cíl usnadnit testování kódu. (Idan 2016)

JavaScript

JavaScript sice může podle názvu působit, jako by byl od Javy odvozen, mají však společný pouze název a trochu podobnou syntaxi. JavaScript je rovněž objektově orinetovaný a je určený pro webové aplikace. V současnosti se nejčastěji využívá pro tvorbu interaktivních webových stránek. Je totiž možné jej psát přímo do HTML kódu (o HTML pojednává kapitola 2.2.7). Jedná se o interpretovaný klientský jazyk, to znamená, že běží přímo uvnitř prohlížeče a jeho překlad probíhá za běhu programu. (Žára 2015) Pro Javascript existuje obrovské množství API (u JavaScriptu jsou označované jako knihovny), mezi nejznámější patří Bootstrap a React. JavaScript může být použit i pro psaní serverových částí aplikace, jednou z technologií, která to umožňuje, je serverový framework Node.js.Jazyk je od roku 1997 standardizován asociací ECMA (European Computer Manufacturers Association). Nejnovější současná verze jazyka je z roku 2016 a může být označována dvěma způsoby: ECMAScript (zkráceně ES) 7 a ES2016. (W3Schools 2018)

C, C++, C#

C je označení pro jeden z nejstarších programovacích jazyků. V současnosti se používá především pro systémový software. C je procedurální kompilovaný jazyk, na kterém je založeno mnoho jazyků v současnosti využívaných, které jsou označovány jako C-like. Příkladem rozšíření jazyka C je jeho objektově orientovaná varianta C++, jejíž název vznikl z původního jazyka C a z operátoru inkrementace „++“, nebo C# (čte se jako C Sharp), který vyvinul Microsoft a využívá platformu .NET Framework. (Holub 2012)

2.2.7 Neprogramovací jazyky

Do neprogramovacích jazyků jsou zahrnovány včechny počítačové jazyky, které neumožňují definici algoritmů. Patří sem především jazyky pro popis struktury dokumentů či pro definování jejich formátu. Tato podkapitola se zaměří na některé neprogramovací jazyky, které jsou v současnosti často využívány.

Standard Generalized Markup Language

Standard Generalized Markup Language (zkráceně SGML) je možné chápat jako standard pro psaní značkovacích jazyků, který umožňuje definovat jiné značkovací jazyky jako podmnožiny jazyka SGML. (Gála 2009)

Extensible Markup Language

Extensible Markup Language (zkráceně XML) slouží k popisu struktury dokumentu. Jeho cílem je umožnit výměnu dokumentů mezi aplikacemi bez nutnosti lidského zasáhu. Řídí se následujícími principy: (Kosek 2000)

- Formát XML musí být použitelný v rámci internetu, otevřený a použitelný kýmkoli. Musí být kompatibilní s jazykem SGML, ze kterého vychází.
- Dokument musí být přizpůsoben pro strojové zpracování a manipulace s ním nesmí být příliš složitá. Měl by být zároveň napsaný tak, aby byl srozumitelný i člověku.
- Množství variant jazyka by mělo být minimální, nepodporuje se tvorba dialektů.
- Struktura dokumentu se skládá z několika částí. (Kosek, 2000)
- *Prolog* je úvodní část dokumentu, která definuje verzi jazyka, kódování a další informace o dokumentu.
- Dále je dokument složen z *elementů*, které jsou do sebe vzájemně vnořené. Pouze jeden z nich může být kořenový. Element se v textu vyznačuje pomocí takzvaných tagů. Většinou se jedná o tagy párové, jeden počáteční a jeden ukončovací, které ohraničují obsah elementu. Názvy tagů se zapisují mezi znaky „<“ a „>“. Ukončovací tag má před svým názvem ještě znak „/“, aby se snadno odlišil od počátečního. Pokud element nemá žádný obsah, je možné využít zkrácený zápis, za název počátečního tagu se před ukončovací špičatou závorkou píše navíc symbol „/“, ukončovací tag se potom nepoužívá.

- Elementům je dále možné v počátečním tagu definovat *atributy*. Atributy se zapisují mezi název a ukončovací závorku. Od názvu elementu a od sebe navzájem se oddělují mezerami. Atribut je složen z jeho názvu, symbolu „=“, a hodnoty ohraničené uvozovkami.

Hypertext Markup Language

Hypertext Markup Language (zkráceně HTML) je další typ značkovacího jazyka odvozeného od SGML. Používá se především pro popis struktury webových stránek. Jeho struktura je podobná jako u jazyka XML, nepodporuje však na rozdíl od XML použití vlastních tagů. HTML definuje strukturu webové stránky a význam jednotlivých částí. Pro specifikaci vzhledu jednotlivých elementů je sice do jisté míry možné použít jejich atributy, vhodnější je však použít kaskádové styly, které poskytují mnohem více možností. Jeho poslední stabilní verze je označovaná jako HTML5. Některé prvky tohoto jazyka nejsou podporovány všemi prohlížeči, především u jejich starších verzí může docházet k problémům. Proto je vhodné si vždy ověřit, jaké prohlížeče jsou schopny daný element zobrazit, a případně hledat jiné lépe podporované alternativy. (Goldstein 2015)

Cascade Style Sheets

Cascade Style Sheet (zkráceně CSS) je označení pro jazyk, který slouží k definování stylu jednotlivých elementů jazyka HTML. Pomocí CSS se popisuje, jakým způsobem je element zobrazen, definovat lze nejrůznější vlastnosti od pozice elementu přes jeho velikost, barvu, průhlednost, pořadí, zaoblení rohů, použitý font až po možnosti animace. Styly mohou být defnované přímo v HTML elementech v atributu style, ve hlavičce HTML dokumentu či v odděleném souboru (tento postup je preferován). Nejnovější stabilní verze jazyka je CSS3. Existují různé nástroje, které umožňují usnadnění zápisu CSS, například Less.js, který je nadstavbou pro CSS a umožňuje používat některé prvky programovacích jazyků, jako například definování proměnných či uplatnění dědičnosti. (Goldstein 2015)

JavaScript Object Notation

JavaScript Object Notation (zkráceně JSON) je spíš datovým formátem než jazykem, je však vhodné ho zde také zmínit. Slouží totiž stejně jako značkovací jazyky k popisu struktury dokumentu. Je objektově orientovaný, platformově nezávislý, a jak již vyplývá

z jeho názvu, je založen na JavaScriptu. Data jsou ukládána v polích či asociativních polích (místo pořadí hodnoty v poli je hodnota reprezentována unikátním klíčem). Ty lze navíc do sebe vnořovat. Povolené hodnoty jsou textové řetězce, čísla a speciální hodnoty true/false/null. (Žára 2015)

2.2.8 Databáze

Různé typy dat (obrazová, textová, zvuková,…) je třeba nějakým způsobem prezentovat, uchovávat a zpracovávat.

Nejmenší jednotka pro uchovávání dat je *bit*. Může nabývat pouze dvou hodnot, nejčastěji jsou tyto dvě hodnoty reprezentovány jako nuly a jedničky. Skupina osmi bitů je označována jako *byte*. Sekvence bitů či bytů potom slouží k zápisu jednotlivých *znaků*. Znaky jsou dále seskupovány do většího celku označovaného jako *položka* (field). Ta slouží k popisu vlastnosti nějakého objektu. Skupina položek, které popisují určitý objekt, je nazývána *záznamem* (record). Související záznamy jsou potom seskupovány do *souborů dat* (file). Každý záznam je potom v souboru identifikován buď svým pořadím, nebo *klíčem*, kterým je určitá položka záznamu, která ho dokáže jednoznačně identifikovat. Nejvyšší úrovní je potom *báze dat* (database). V ní jsou seskupovány různé soubory dat. (Gála 2009)

Přístup k ukládání a správě dat lze rozdělit na tradiční a databázový. (Gála 2009)

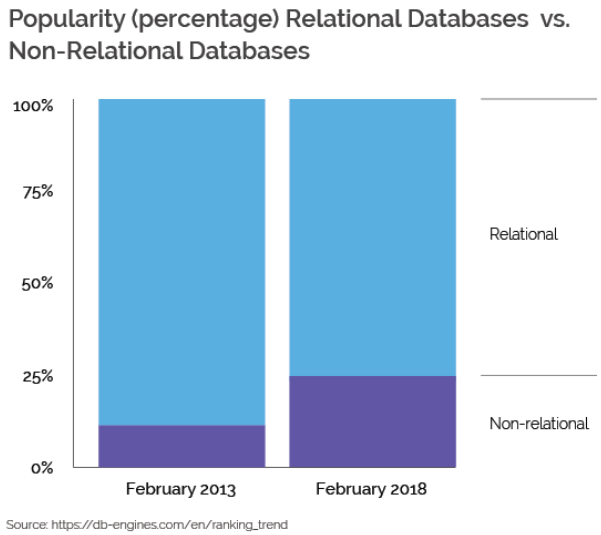
- *Tradiční* přístup je založen na využití klasického souborového systému pro vytváření, čtení, úpravu a mazání souborů. Báze dat je tvořena samostatnými soubory, se kterými pracují různé aplikace.
- Druhý přístup se označuje jako *databázový*. Soubor dat je označován jako entita, jednotlivé záznamy jsou výskytem entity a položky jsou atributy dané entity. Báze dat je označována jako databáze. Ta je dále doplněna systémem řízení báze dat (zkráceně SŘBD) a společně tvoří databázový systém. Pro definování struktury dat se používá *jazyk definice dat* (Data Definition Language, zkráceně DDL). Společně s databází vzniká také *datový slovník* (Data Dictionary) v němž se shromažďují informace o databázi. Databázový systém zahrnuje i jazyk manipulace s daty (Data Manipulation

Language, zkráceně DML), který slouží k přístupu k uloženým datům, a jazyk pro řízení dat (Data Control Language, zkráceně DCL). V současnosti se používají především relační a objektové databáze. (Pavlíček 2012)

Proces návrhu databáze se nazývá datové modelování. Je součástí fáze návrhu aplikace. V této fázi se tvoří modely, které mají usnadnit implementaci systému. Návrh datového modelu je možné rozdělit do tří úrovní podrobnosti: (Gála 2009)

1. *Konceptuální* návrh slouží k základnímu návrhu entit a jejich vazeb. Zaměřuje se na strukturu dat a nezohedňuje fyzické charakteristiky databáze.
2. *Logický* návrh je o něco podrobnější, slouží k návrhu struktury již pro konkrétní databázový systém. Zahrnuje specifičtější popis entit včetně jejich atributů.
3. *Fyzický* návrh definuje fyzický datový model, který obsahuje všechny podrobnosti, které jsou potřebné pro implementaci.

Na následujícím obrázku je možné vidět porovnání oblíbenosti relačních a nerelačních databází v únoru roku 2013 a v únoru roku 2018.



Obrázek 15: Porovnání oblíbenosti relačních a nerelačních databází
Zdroj: Hammink, 2018

Relační databáze

Principy relačních databázi formuloval E. F. Codd v roce 1970. Soubor dat je nazýván relací, která má formu tabulky, její řádky jsou jednotlivé záznamy a každý sloupec představuje jednu položku. (Gála 2009)

S relačními databázemi souvisí pojem integrita, kterou je možné vysvětlit jako celistvost, soudržnost a neporušenost dat. Rozlišovány jsou tři druhy integrity: entitní, doménová a referenční. (Gála 2009)

- *Entitní* integrita zajišťuje, že každý záznam v relaci má unikátní jednoznačný identifikátor, který je označován jako *primární klíč*.
- *Doménová* integrita zajišťuje, aby každý atribut byl vybrán pouze z množiny přípustných hodnot.
- *Referenční* integrita popisuje způsob definování vazeb jednotlivých relací. Pokud má mít jedna relace odkaz na jinou relaci, je třeba do ní přidat další atribut, který je označován jako *cizí klíč*. Do tohoto atributu se potom pro každý záznam uloží hodnota, která slouží jako odkaz na jinou relaci. Takovou hodnotou je pak obvykle primární klíč záznamu v jiné relaci. Referenční integrita zajišťuje, že je jako cizí klíč možné použít pouze hodnoty, které jsou v dané relaci použity jako primární klíče. Zároveň také neumožní odstranění záznamu, na který se pomocí cizího klíče odkazuje jiná relace.

Existuje mnoho různých databázových systémů od různých výrobců. Proto byl zaveden jazyk SQL (Structured Query Language), který má za cíl příkazy pro přístup k databázi standardizovat. (Pavlíček 2012) Jednotlivé příkazy tohoto jazyka lze rozdělit do čtyř skupin: (Gála 2009)

- Příkazy pro manipulaci s daty (DML) zahrnují příkaz SELECT, který slouží k výběru dat z databáze, příkaz INSERT, který slouží ke vložení nových dat, příkaz UPDATE umožňuje jejich úpravu a DELETE jejich smazání.
- Příkazy pro definici struktury databáze (DDL) slouží k úpravě struktury databáze. Příkaz CREATE slouží k vytvoření nové relace, příkaz ALTER relace umožňuje upravovat a DROP mazat.

- Příkazy pro řízení dat (DCL) slouží k řízení přístupu k datům. Zahrnují příkaz GRANT, který uživateli přidělí zvolená přístupová práva k databázi či její části, REVOKE umožňuje práva uživateli odebrat, BEGIN vymezuje začátek transakce, COMMIT její potvrzení a ROLLBACK slouží k vrácení změn od založení transakce, pokud během ní dojde k chybám.
- Ostatní příkazy se zaměřují na ostatní funkce, například definování uživatelů, kódování znaků atd.

Nerelační databáze

Nerelační databáze, také označované jako NoSQL (která ale neodmítá použití SQL, zkratku lze vysvětlit jako Not Only SQL – nejen SQL, nevylučuje použití jazyka SQL) databáze, jsou založené na trochu jiné architektuře než databáze relační. Druhy nerelačních databází je možné rozdělit do pěti následujících skupin: (Hammink 2018)

- *Key-value* – jednoduchá forma databáze, která slouží k ukládání dvojic klíč – hodnota. Používá se v případech, kdy je třeba přistupovat rychle k datům, které nevyžadují komplexnější struktury. Příkladem takové databáze je MemcacheDB.
- *Wide column* – databáze podobné relačnímu modelu, ale každý řádek reprezentující záznam v databázi může mít odlišný počet sloupců a typ jejich hodnot. Obvykle se využívá pro velké objemy dat s vysokým množstvím atributů. Příkladem může být Google Cloud Bigtable.
- *Document-oriented* – tyto databáze umožňují uchovávat data ve formě dokumentů. Formát dokumentu se liší dle zvoleného databázového systému, může se jednat o formát XML, JSON nebo například soubory aplikace Microsoft Excel. Příkladem takové databáze je MongoDB, která data ukládá pomocí formátu JSON.
- *Graph-oriented* – jsou založeny na flexibilním grafickém znázornění uložených dat. Každý objekt je reprezentován pomocí uzlu a vazby mezi objekty jsou reprezentovány pomocí hran, které je spojují. Příkladem grafové databáze je Neo4j od stejnojmenné společnosti.

2.2.9 Business Intelligence

Business Intelligence (zkráceně BI) je možné chápat jako ucelený a efektivní přístup k práci s firemními daty, který zahrnuje sadu procesů, know-how, aplikace i technologie a má za cíl usnadnit strategická rozhodnutí a tím ovlivnit úspěch společnosti. Informovanost totiž znamená jednu z hlavních konkurenčních výhod a business intelligence může informovanost výrazně zvýšit. (Panec 2003)

Základem business intelligence je transformace zdrojových dat na znalosti užitečné pro firmu. Zdrojová data jsou označována jako Big Data. Jedná se o data takových rozměrů, že je běžené editory, jako například Microsoft Excel, nedokáží efektivně zpracovat. Celý proces je označován jako Intelligence Value Chain neboli řetězec zkvalitnění znalostí. Zahrnuje následující kroky: (Panec 2003)

- Plánování je prvotní část procesu a slouží k přípravě akčního plánu, který stanovuje strategii, jak dosáhnout požadovaných cílů.
- Dalším krokem je extrakce, transformace a načtení dat (*Extract-Transform-Load*, zkráceně ETL). Jak již z označení fáze vyplývá, obsahuje tři kroky:
    - Extrakce dat z různých zdrojových systémů je prvním krokem. Vyberou se pouze ta data, která má smysl dále analyzovat pro požadovaný cíl.
    - Druhým krokem je transformace, která zahrnuje akce, které vedou k převedení dat do podoby potřebné pro jejich uložení a další zpracování.
    - Posledním krokem je načtení dat do analytických databází. Těmi jsou obvykle *datové sklady* (data warehouse), které uchovávají data ze všech oblastí, a *datová tržiště* (data mart), která jsou zaměřena obvykle jen na jednu konkrétní oblast. Data jsou obvykle uložena v multidimenzionálních modelech, protože se očekává, že zkoumaná skutečnost závisí na více než jednom faktoru.
- Další fází je potom inteligentní skladování obvykle obrovského množství dat. Je třeba zvolit vhodnou strukturu, spolehlivé technologie a vhodné nástroje pro přístup k datům. O způsobu uchování dat se obvykle rozhoduje už ve fázi plánování.
- Čtvrtá část procesu potom zahrnuje přímo postupy business intelligence k efektivnímu získávání znalostí z dat uložených v datových skladech a tržištích. Při práci

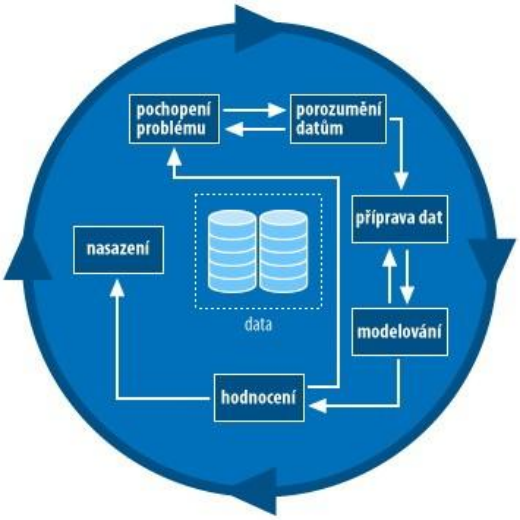
s multidimenzionálními daty se obvykle využívají technologie označované zkratkou OLAP (Online Analytical Processing), které poskytují efektivní přístup k datům.

- Pátá fáze je určitou nadstavbou, která zahrnuje některé další analytické procesy, například predikce budoucího stavu či hledání skrytých souvislostí.

Business intelligence je velmi obsáhlá oblast informatiky, uplatňuje se však především u rozsáhlých informačních systémů, proto se na ni tato práce zaměřila jen okrajově. Navíc business intelligence spadá do procesu vývoje softwaru jen částečně, může se jednat například o návrh databází tak, aby následné procesy business intelligence měly k dispozici všechna důležitá data.

2.2.10 Data Mining

Data Mining, česky označované jako dolování dat, je „soubor metod sloužících ke zpracování dat a získání netriviálních informací, které jsou v nich obsažené“ (Procházka 2017). Data mining je složitý proces pro hledání skutečností ukrytých v datech. Proces je možné rozdělit do dvou skupin – *predikce* a *deskripce*. Predikce je možné chápat jako předpovědi vyvozené na základě zdrojových dat a deskripce je převedení predikce do formy, které porozumí i neodborná veřenost. Cílem této práce není zkoumat podrobnosti související s dolováním dat, proto průběh procesu bude znázorněn pouze pomocí následujícího obrázku a autorka této práce nepovažuje za nutné ho podrobněji rozebírat. (Procházka 2017)



Obrázek 16: Data Mining
Zdroj: Procházka, 2017

2.3 Řízení vývoje SW

Řízení vývoje softwaru vychází především z teorie projektového managementu. Vývoj softwaru totiž v hodně ohledech připomíná projekt. Ve vývojovém týmu musí být někdo, kdo za řízení vývoje bude mít zodpovědnost a bude k tomu mít také odpovídající pravomoce. Úkolem takové osoby je potom zajistit, že software bude dodán v domluvené kvalitě, přestože je jeho vývoj omezen plánem, a to především termínem a rozpočtem. Protože je jeho funkce podobná roli projektového manažera, bude pro takovou osobu v této práci používáno právě označení *projektový manažer*. Jeho cíle jsou následující: (Sommerville 2013)

1. Dodat software v domluveném termínu.
2. Udržet celkové náklady tak, aby nepřesáhly rozpočet.
3. Dodat takový software, který zákazník požaduje.
4. Udržet dobře fungující vývojový tým se spokojenými členy.

2.3.1 Produkt

Produkt je výsledek nějakého procesu. Software je produktem, ale oproti hmotným statkům má jisté odlišnosti. Software je nehmotný logický konstrukt a nemůže podléhat

opotřebení ani vadám materiálu. Chyby jsou nejčastěji způsobeny návrhem či implementací. Požadavky uživatelů se mohou změnit i v krátkém časovém horizontu a na začátku vývoje často nejsou známy či jsou popsány nedostatečně. Hardware i software se dynamicky vyvíjí a prochází rychlou obměnou. (Roudenský 2016)

2.3.2 Projekt

Vývoj software lze v mnoha ohledech chápat jako realizaci projektu. Projekt je chápán jako *"jedinečný proces sestávající z řady koordinovaných a řízených činností s daty zahájení a ukončení, prováděný pro dosažení cíle, který vyhovuje specifickým požadavkům, včetně omezení daných časem, náklady a zdroji"* (Komzák 2013, s. 23). Při vývoji softwaru je vývojový tým rovněž omezen rozsahem a termínem, kterým se zavázal zadavateli, rozpočtem, který je sponzor ochotný vynaložit, a především kvalitou, aby byl pro cílové zákazníky přijatelný. Na vývoji software pracuje obvykle tým osob a stejně tak jako projekty musí být vývoj software někým řízen a organizován, a to jak po časové, tak po nákladové stránce. Velmi důležitá je jasně stanovená dělba práce v týmu. Proto je pro organizaci vývoje vhodné používat nástroje a techniky projektového řízení. (Avison 2009)

Třem omezením – rozsahu, termínu a rozpočtu – se v oblasti projektového řízení říká *projektový trojimperativ*. Kvalita se potom odvíjí od trojimperativu. Pokud je rozsah přiměřený termínu a rozpočtu, potom je možné vyvíjet kvalitní software. Tradiční přístup předpokládá, že rozsah, termín a rozpočet jsou sjednány na začátku a nelze je měnit. Potom je jedinou proměnnou kvalita, která se snižuje s každou neočekávanou změnou, která vede k navýšení nákladů či práce. Agilní přístup naopak má za cíl vytvářet co nejkvalitnější software a v případě neočekávaných problémů tým se zadavatelem jedná o tom, zda dojde k posunutí termínu, navýšení rozpočtu či snížení rozsahu, kvalita by se snižovat ale neměla. (Kadlec 2004) Přesnější popis těchto přístupů je součástí kapitoly 2.5.

2.3.3 Správa rizik

Správa rizik představuje jeden z hlavních úkolů projektového manažera. Zahrnuje aktivity, které slouží k předpovídání a eliminaci rizik. Rizika je možné rozdělit do tří kategorií: (Kadlec 2004)

- *Projektová rizika* jsou rizika, která ovlivňují plán projektu či jeho omezení, jako je termín a rozpočet.
- *Technická rizika* mají vliv na výkon softwaru či jeho kvalitu.
- *Obchodní rizika* ovlivňují organizaci, která software vyvíjí nebo si ho pořizuje. Souvisí s trhem, na kterém je produkt nabízen, a s dalšími subjekty, které na tomto trhu působí.

Proces správy rizik lze rozdělit do čtyř základních fází: (Sommerville 2013)

1. Identifikace rizik je první fází procesu a má za cíl najít potenciální projektová, produktová a obchodní rizika. Vhodným nástrojem pro zahájení této fáze je brainstorming, do kterého se mohou zapojit všichni členové vývojového týmu, případně i některé externí role, jako je zadavatel či uživatel. Během brainstormingu zúčastněné osoby navrhují možná rizika, bez ohledu na to, jak pravděpodobný je jejich výskyt. Výstupem této fáze by měl být přehled všech rizik, která mohou nastat.
2. Druhá fáze, označovaná jako analýza rizik, slouží k podrobnějšímu prozkoumání rizik nalezených během první fáze. U každého rizika by měla být posouzena jeho závažnost a pravděpodobnost. Jedná se o subjektivní fázi, jejíž úspěšnost je založena na zkušenostech osob, které analýzu provádějí. Pravděpodobnost ani závažnost rizik není možné přesně vyhodnotit, obvykle se pro ně využívají číselné stupnice nebo slovně vyjádřené úrovně. Výstupem této fáze by měla být sada rizik, která je možné řadit podle pravděpodobnosti i podle závažnosti.
3. Třetí fáze slouží k plánování, jak rizikům zabránit. Pro klíčová rizika, která byla nalezna a analyzována během předchozích fází, jsou nyní definovány strategie, které by jejich výskyt měly omezit. Také se během této fáze definují postupy pro měření výskytu rizik. Strategie pro zvládání rizik lze rozdělit do tří skupin:

    - *Preventivní* strategie mají za cíl riziku předejít.

- *Minimalizační* strategie slouží k minimalizaci dopadu rizika, pokud nastane.
- Při použití *kontingenční* strategie jsou připravena řešení pro nejvážnější rizika, která mohou nastat.

4. Obsahem čtvrté fáze je rizika monitorovat a hodnotit. Pokud některá z rizik nastanou, je možné ověřit, zda byl odhad jejich pravděpodobnosti či následků správný, a zhodnotit, jaký efekt měla preventivní opatření přijatá během třetí fáze.

2.3.4 Řízení týmu

Řízení týmu spočívá nejen v rozdělování práce mezi jednotlivé členy vývojového týmu. Práce členů týmu musí být také nějakým způsobem kontrolována, a pokud jsou odhaleny nedostatky, je třeba řešit, jak je napravit, aniž by to ohrozilo plán projektu.

Osoby, které se na vývoji podílejí, je potřeba motivovat, aby prováděly lepší práci a aby vůbec měly chuť v práci pokračovat. Je praxí prokázané, že mzda sama o sobě nemusí stačit jako dlouhodobý motivátor k lepším výkonům, je tedy třeba posuzovat postoj k práci každého člena zvlášť a najít pro něj vhodnou motivaci. (Avison 2009)

Projektový manažer tedy musí mít kromě znalostí a zkušeností z oblasti softwarového inženýrství také schopnosti pro řízení lidí. Projektový manažer by měl vycházet se všemi členy týmu, ale zároveň by jimi měl být respektován. Projektový manažer musí dohlížet také na týmovou práci a na případné neshody mezi jednotlivými členy týmu. Také by měl zajistit plynulost komunikace uvnitř týmu, díky níž budou mít jednotliví členové přístup ke všem informacím, které potřebují ke své práci. (Sommerville 2013)

Velmi důležitá je také hierarchie a uspořádání skupiny. Manažer může některé ze svých povinností delegovat. Například může zvolit osobu, která bude dohlížet na kvalitu kódu, komunikaci s uživateli, analýzu rizik či jiné oblasti vývoje softwaru. Musí však dané osobě také zajistit prostředky, pomocí kterých může dosáhnout cílů, které jsou od ní požadovány. Neměla by existovat oblast vývoje softwaru, za kterou není nikdo zodpovědný.

2.3.5 Řízení znalostí

Cílem řízení znalostí je spojit ty, kteří mají určitou znalost, s těmi, kdo ji potřebují. Umožňuje proměnit znalosti jednotlivců ve znalosti týmu či organizace. Zahrnuje nejen sepisování, uchovávání a přenos znalostí uvnitř týmu, ale také proškolování pracovníků či workshopy. (Gála 2009)

2.4 Proces vývoje SW

Softwarový proces je sekvence aktivit, které vedou ke vzniku softwaru. Softwarové procesy zahrnují technické, kolaborativní a manažerské aktivity s cílem navrhnout, implementovat a otestovat softwarový produkt. Je možné je shrnout do 5 hlavních aktivit, které budou popsány v následujících podkapitolách:

1. Specifikace
2. Návrh
3. Vývoj
4. Testování
5. Provoz

2.4.1 Specifikace softwaru

Specifikace softwaru umožňuje určit a definovat, kterými funkcemi má software disponovat. Jedná se o jednu z nejdůležitějších aktivit, chyby v této fázi vedou k problémům v následujících fázích. Čím později je chyba objevena, tím více nadbytečných nákladů je třeba vynaložit na její opravu. (Kadlec 2004)

Výstupem této fáze je dokument s odsouhlasenými požadavky, které vyhovují všem zúčastněným stranám. Požadavky jsou sepsány obvykle ve dvou formách zároveň. Zjednodušená forma slouží pro snazší pochopoení zadavatelem a uživateli. Podrobnější forma je potom určena pro vývojáře, kteří potřebují dodatečné informace, aby bylo možné funkčnosti implementovat. (Kadlec 2004)

Specifikace softwaru se obvykle skládá ze čtyř dílčích aktivit, které se mohou vzájemně prolínat: (Sommerville 2013)

- *Zjišťování a analýza požadavků* spočívá v diskuzi s potenciálními uživateli aplikace, zadavatelem a dalšími zúčastněnými stranami a hledání jejich potřeb.
- *Studie proveditelnosti* obnáší zpracování odhadu, který stanoví, zda je možné vyhovět stanoveným požadavkům pomocí aktuálně dostupných softwarových a hardwarových možností. Zkoumá se také ekonomická efektivita pro vývojový tým a různá omezení, která na vývoj mohou mít vliv, především termín a rozpočet. Studie proveditelnosti by měla být provedena v samém začátku procesu vývoje, ale je nutné mít alespoň základní představu o plánované aplikaci.
- Během *specifikace požadavků* jsou poznatky získaté v předchozích dílčích aktivitách sepsány obvykle ve dvou úrovních podrobnosti – pro uživatele a pro vývojáře. Výstupem je dokument se sadou požadavků.
- *Validace požadavků* slouží ke kontrole, zda jsou definované požadavky realistické, konzistentní a úplné. Během této fáze by měly být odstraněny všechny nalezené nedostatky, aby nezpůsobovaly problém při návrhu, implementaci či pozdějších fázích vývoje.

2.4.2 Návrh softwaru

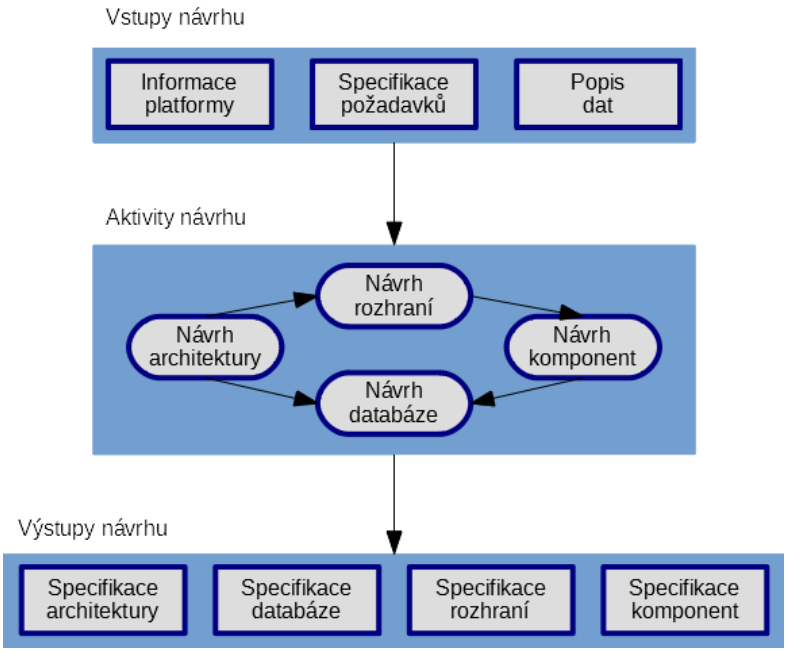
Návrh softwaru slouží k popisu vyvíjené aplikace. V této fázi se definuje použitá softwarová platforma, požadavky na hardware, aplikační architektura, doporučené algoritmy, struktura aplikace a vazby mezi jednotlivými částmi, popis jednotlivých funkčností včetně základního a alternativních scénářů, datový model, případně rozhraní aplikace pro komunikaci s aplikacemi třetích stran. (Kadlec 2004)

Aktivity, které jsou součástí návrhu aplikace, se různí podle toho, jaký software je vyvíjen. Během návrhu běžné aplikace je většinou potřeba provádět tyto aktivity (Sommerville 2013):

- *Návrh architektury*, během kterého jsou definovány základní komponenty softwaru a vzájemné vazby mezi nimi.

- *Návrh rozhraní*, které spočívá v definování rozhraní k jednotlivým částem aplikace. S dobře definovaným rozhraním je možné napojovat na aplikaci jiné moduly či aplikace, aniž by bylo třeba znát její zdrojový kód.
- *Návrh komponent*, což je vlastně popis předpokládaných implementovaných funkcí a vzhledu jednotlivých částí.
- *Návrh databáze*, který obnáší návrh datového modelu, jednotlivých entit, jejich atributů, které je třeba ukládat, datových formátů a dalších požadavků na datové úložiště.

Není nutné vždy provádět všechny aktivity. Pokud aplikace nevyžaduje datové úložiště, samozřejmě není nutné navrhovat databázi. Na obrázku níže jsou znázorněny klíčové aktivity, vstupy a výstupy fáze návrhu. Některé aktivity mohou být prováděny současně, je však důležité je po dokončení zhodnotit s ohledem na vzájemné vazby a konzistenci vyvíjeného software. (Sommerville 2013)



Obrázek 17: Postup návrhu
Zdroj: Sommerville 2013

2.4.3 Vývoj softwaru

Vývoj software probíhá na základě zpracovaného návrhu. V některých případech se fáze návrhu a vývoje prolínají a návrh se sepisuje dynamicky v průběhu vývoje. Většinou ale

platí pravidlo, že čím lepší a propracovanější je návrh, tím snazší a plynulejší bude vývoj. Návrh a vývoj aplikace lze do určité míry automatizovat, existují nástroje, které dle návrhu vygenerují základní strukturu programu (například založí třídy a konstanty, předpřipraví metody s parametry, návratovou hodnotou, komentáři), vývojář se potom stará jen o psaní kódu a nemusí řešit přepisování informací z návrhu. Samotný proces programování je potom v režii vývojářů a velmi důležitá je vhodná dělba práce uvnitř vývojového týmu. (Kadlec 2004)

2.4.4 Testování softwaru

Fáze testování slouží k ověření, že software vyhovuje své specifikaci a že splňuje očekávání uživatelů. (Kadlec 2004) Testování spočívá ve spouštění funkčností aplikace s testovacími daty. Připraví se taková data, aby bylo možné otestovat jak hlavní scénář, tak všechny očekávané alternativní scénáře. Je také důležité otestovat, jak bude software reagovat na různé neočekávané typy vstupních dat. Fáze testování může zahrnovat také revize kódu pro ověření jeho kvality.

Testování se často prolíná s dalšími aktivitami vývojového procesu. Software je vhodné začít testovat co nejdříve od jednotlivých komponent, aby případné chyby mohly být opraveny již během vývoje. Testování by mělo zahrnovat tyto aktivity: (Sommerville 2013)

- *Vývojové testování* zahrnuje testování jednotlivých komponent vývojáři během vývoje. Většinou se pro tyto účely píší jednotkové testy, které automaticky spouští jednotlivé funkčnosti se zadanými vstupními daty.
- *Integrační testování* slouží k odhalení chyb, které vznikly po sjednocení všech komponent do softwarového produktu. Systém je testován jako celek včetně vzájemné komunikace komponent. Za předpokladu nedostatečně podrobného návrhu či většího vývojového týmu, kde každou komponentu tvoří jiný vývojář, může docházet k neočekávaným problémům při interakci komponent. Jejich příčiny mohou souviset s odlišnými zvyklostmi vývojářů, například rozdíly v použitém kódování či formátu dat. Proto je velmi důležitý návrh, tyto chyby se většinou objeví až v posledních fázích

vývojového procesu a jejich náprava vede ke značným dodatečným nákladům. Po opravení chyby je totiž potřeba, aby software znovu prošel zbytkem životního cyklu.

- *Předávací testování* je závěrečnou částí testovací fáze. Software se testuje s daty, které dodal zadavatel či uživatelé, případně software mohou testovat přímo oni v testovacím prostředí. Předávací testy mohou pomoci odhalit určitá opomenutí z předchozích fází, testovací data totiž obvykle definuje stejný tým, který software vyvíjí, proto je vhodné software otestovat s daty od osob, které budou software používat. Předávací testování lze rozdělit do dvou částí: alfa a beta testování.
    - *Alfa testování* je akceptační testování zadavatelem, který ověřuje, zda software vyhovuje jeho požadavkům a případně zadá vývojovému týmu změnové požadavky.
    - *Beta testování* spočívá ve spuštění beta verze programu veřejnosti k užívání, obvykle zdarma. Uživatelé získají možnost si software vyzkoušet a vývojáři získají zpětnou vazbu, mohou se odhalit některé další nedostatky a chyby. Rovněž se ověří, jak bude software reagovat na vyšší zátěž.

2.4.5 Provoz softwaru

Fáze provozu začíná ve chvíli, kdy je nasazena první stabilní verze do produkčního prostředí a běžní uživatelé ji mohou začít používat. Obvykle je fáze provozu nejdelší fází procesu vývoje software.

V této fázi by již měla být většina chyb odhalena, i přesto se ale může stát, že bude třeba něco opravit, například pokud uživatel zadá vstup, se kterým nikdo nepočítal a který způsobí havárii softwaru. Tomu lze snadno předejít zapojením uživatele do fáze testování. Dalším problémem, který je obvykle odhalen až během provozu, jsou chyby způsobené přetížením softwaru větším množstvím uživatelů.

Modely softwarových procesů se různí především v otázce, co se děje se softwarem po spuštění běžného provozu. Je nepravděpodobné, že po nasazení softwaru skončí veškerá práce na něm. Po nasazení je obvykle potřeba buď opravit nově nalezené chyby, přidat nové funkčnosti či modifikovat ty stávající, software je nějakým způsobem dále rozvíjen. Některé modely změny netolerují s myšlenkou, že dobře navrženou a implementovanou

aplikaci upravovat není třeba. Jiné modely zohledňují rozvoj software jako další fázi vývojového procesu. Proces rozvoje software může být zkoumán také odděleně od procesu vývoje s některými podobnými a některými odlišnými znaky. Rozvoj software však může být chápán také jako vznik nové verze, která opět prochází všemi fázemi vývojového procesu, a právě tomuto pojetí dává autorka této práce přednost. Nezáleží na tom, zda se jedná o prvotní verzi či několikrát upravenou, jednotlivé fáze jsou velmi důležité především z hlediska spolehlivosti aplikace, proto by se od nich nemělo upouštět jen proto, že je potřeba provést v aplikaci pouze drobné změny.

Existují dva odlišné přístupy k prevenci nadbytečných nákladů vzniklých vlivem zapracování změn: (Sommerville 2013)

- *Prevence změn* je přístupem, který se snaží nákladům na přepracování softwaru předejít. Zahrnuje aktivity, které mají za cíl co nejlépe zanalyzovat potřeby uživatelů tak, aby nebylo nutné software měnit v průběhu provozní fáze. Tento přístup je typičtější u tradičních metodik.
- *Tolerance změn* očekává, že bude potřeba software měnit, a má za cíl minimalizovat náklady na zapracování změn. Tomuto cíli by potom měl být přizpůsoben proces vývoje softwaru. Tento přístup častěji uplatňují agilní metodiky.

2.5 Modely softwarových procesů

Existuje mnoho modelů softwarových procesů, které lze při plánování vývoje softwaru využít. Každý z následujících modelů definuje proces vývoje software trochu jinak. Model softwarových procesů slouží k popisu aktivit včetně vazeb mezi nimi. Kromě aktivit samotných některé modely zahrnují také produkty, které jsou výsledkem některé z aktivit, role, které jsou za daný proces zodpovědné, či předběžné a následné podmínky pro každou z aktivit procesu. Každý z modelů zobrazuje procesy z určité perspektivy, proto informace, které poskytují, nemusí být kompletní. Některé z modelů se zaměřují na celý životní cyklus softwaru, jiné jen na jeho části.

Modely lze rozdělit do dvou skupin podle toho, z jakého přístupu vycházejí. Existují dva hlavní přístupy – tradiční a agilní – a budou dále popsány v rámci této podkapitoly.

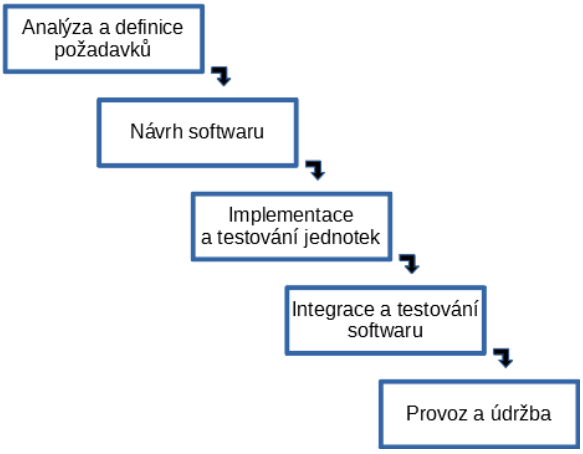
2.5.1 Tradiční přístup

Plánovaný vývoj se odvíjí od teorie projektového řízení. Nejdůležitější je plán, který ovlivňuje úspěch celého projektu. Tyto metodiky jsou i v současnosti využitelné, především pro kritické bezpečnostní systémy. U tradičního vývoje se předpokládá, že domluvený rozsah funkcionalit je sjednán na začátku a nelze jej měnit, od něj se potom odvíjí termín a náklady. Neočekávané změny za předpokladu, že není možné změnit termín ani náklady, mohou být provedeny jen na úkor kvality.

Vodopádový model

Vodopádový model klíčové aktivity rozděluje do pěti kaskádovitě uspořádaných fází procesu. Výsledkem každé z těchto fází by měl být nějaký dokument, což usnadní vedoucím pracovníkům sledování procesu a jeho aktuálního stavu. Nevýhodou tohoto modelu je jeho nepružnost, aktivity jsou striktně rozděleny do pěti fází a rozhodnutí provedená v jedné z fází jsou neměnná pro všechny následující fáze. Jednotlivé fáze by se neměly vzájemně prolínat. Tento model je vhodné využít v případě, že jsou požadavky dostatečně popsány a nepředpokládá se, že se budou v průběhu vývoje měnit. (Kadlec 2004)

V první fázi nazvané „analýza a definice požadavků“ se navrhují základní funkčnosti a vlastnosti softwaru. Ve druhé fázi „návrh softwaru“ je definována softwarová architektura a její klíčové prvky. Třetí fáze „implementace a testování jednotek“ zahrnuje vývoj až po jednotkové testy, které slouží k základnímu otestování funkčností jednotlivých částí softwaru. Ve čtvrté fázi „integrace a testování softwaru“ jsou potom jednotlivé části integrovány do celistvé aplikace a důkladně otestovány a to především vůči návrhu připravenému v druhé fázi procesu. Poslední fáze „provoz a údržba“ začíná v momentu nasazení aplikace k běžnému užívání. Obvykle je to nejdelší fáze, která zahrnuje také opravy chyb, které se během testování neodhalily, či jiné provozní zásahy do aplikace. Jednotlivé fáze jsou zachyceny na následujícím obrázku. (Sommerville 2013)

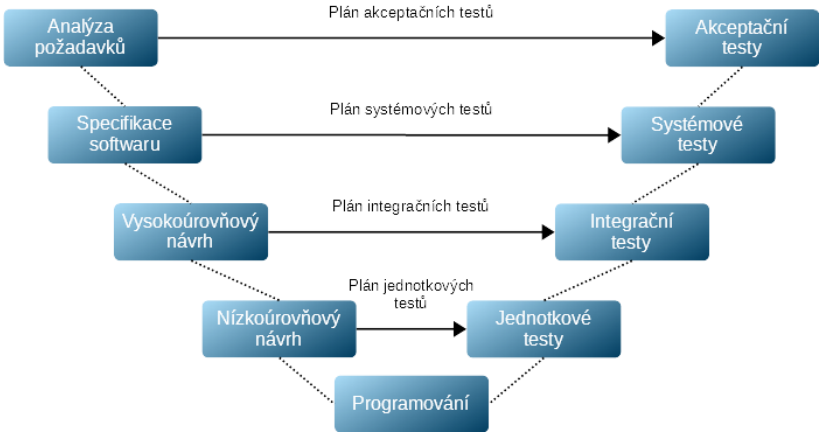


Obrázek 18: Vodopádový model
Zdroj: Sommerville 2013

V-Model

V-Model je založen na vodopádovém modelu. Z něj vychází především levá polovina modelu: sběr požadavků, specifikace produktu, návrh a implementace. Je však rozšířen o několik typů testů, každý z nich ověřuje určitou fázi vývoje (Žoltá 2018):

- *Jednotkové* testy jsou napsány tak, aby je stačilo pouze spustit, a již mají předdefinovaná data. Slouží k testování jednotlivých komponent.
- *Integrační* testy slouží k otestování jednotlivých komponent po jejich integraci do kompletního softwarového produktu.
- *Akceptační* testy jsou poslední fází testů a provádí je buď zákazník, nebo alespoň pro ně definuje vstupní data. Akceptační testy lze rozdělit do dvou druhů (Roudenský 2016):
    - Factory Acceptance Testing (FAT) je označení pro testování, které provádí vývojový tým.
    - User Acceptance Testing (UAT) označuje testování software zadavatelem či koncovým uživatelem. Základním předpokladem je, aby se alespoň jedna ze stran na vývoji podílela.



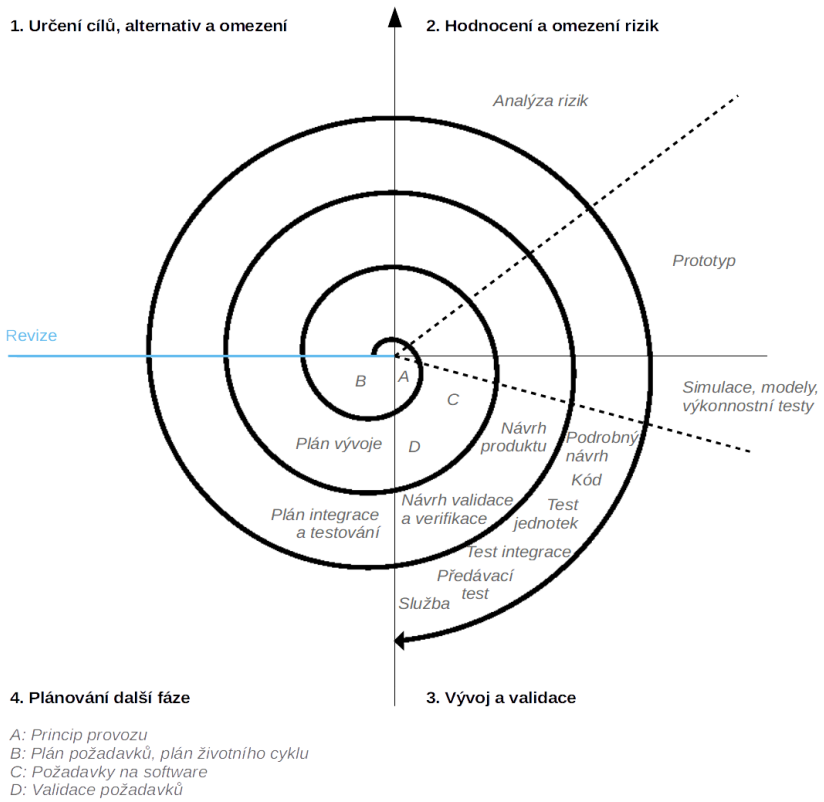
Obrázek 19: V-Model
Zdroj: Žoltá 2018

Spirálový model

Spirálový model je založen na snaze eliminovat riziko, které vede ke změnám v softwarovém produktu. Softwarový proces je reprezentován pomocí spirály, kde každá smyčka představuje jednu fázi procesu. Nepředpokládá se návrat zpět k předchozím fázím, pokud je třeba něco opravit, je třeba projít další smyčkou spirály. (Kadlec 2004)

Každá smyčka ve spirále se dělí na čtyři dílčí kroky: (Sommerville 2013)

1. Prvním krokem je stanovení cílů, kterých by mělo být dosaženo po průchodu smyčkou. Dále se v této fázi definují rizika, omezení a plán řízení vývojového procesu.
2. V dalším kroku jsou nalezená rizika vyhodnocena a následně prováděny akce pro jejich eliminaci.
3. Třetím krokem je vývoj a validace softwarového produktu a jeho jednotlivých částí.
4. V posledním kroku probíhá revize a hledají se možnosti dalšího rozvoje softwarového produktu. Pokud se vývojový tým rozhodne software dále rozvíjet, je v tomto kroku stanoven plán pro další fázi vývojového procesu.



Obrázek 20: Spirálový model
Zdroj: Kadlec 2004

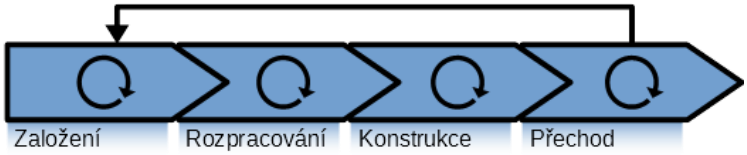
Rational Unified Process

Rational Unified Process, dále RUP, je model založený na modelovacím jazyce Unified Modeling Language, dále UML, který je blíže popsán v kapitole 2.2.3. Snaží se eliminovat nedostatky ostatních modelů, kombinuje jejich prvky, doporučuje postupy pro jednotlivé fáze vývoje softwaru a na vývoj se zaměřuje ze tří pohledů: (Sommerville 2013)

- Dynamická perspektiva znázorňuje fáze modelu v průběhu času.
- Statická perspektiva zobrazuje aktivity vývojového procesu.
- Perspektiva postupů doporučuje optimální postupy pro řešení konkrétních oblastí vývoje softwaru.

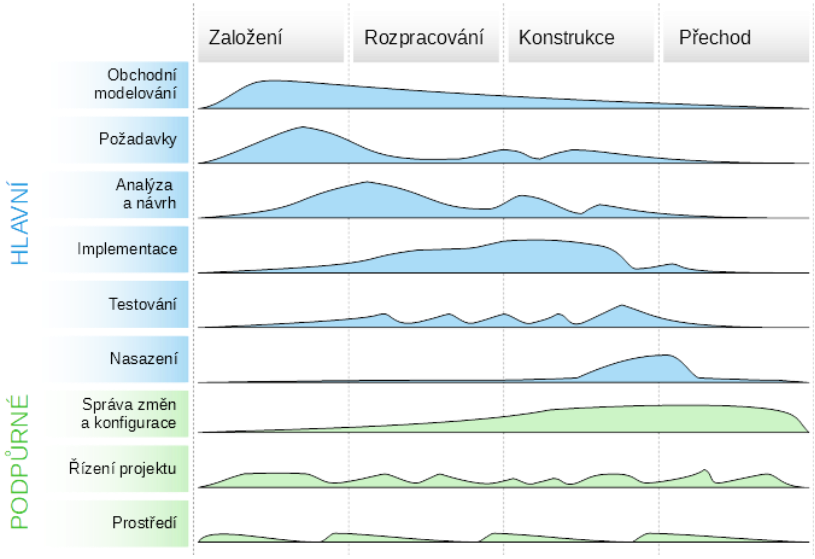
Dynamická perspektiva modelu rozlišuje čtyři fáze softwarového procesu, každou z těchto fází je možné opakovat podle potřeby a stejně tak je možné opakovat celý proces vývoje. (Kadlec 2004)

1. *Založení* (inception) spočívá ve zhodnocení přínosu softwarového produktu a rozhodování, zda má jeho vývoj smysl.
2. *Rozpracování* (elaboration) je fáze, která má za cíl vypracování modelu požadavků na systém, který obvykle obsahuje seznam případů užití, architekturu softwaru, potenciální rizika a plán vývoje softwaru.
3. *Konstrukce* (construction) zahrnuje návrh softwaru, programování a testování. Vyvíjeny jsou nejprve jednotky, které jsou potom integrovány do kompletního softwaru, testování probíhá průběžně.
4. *Přechod* (transition) znamená přenesení systému od vývojové komunity k uživatelské komunitě. Na konci této fáze by měl být spolehlivý software, který správně funguje ve svém provozním prostředí.



Obrázek 21: Model RUP
Zdroj: Sommerville 2013

Statická perspektiva modelu RUP se zaměřuje na pracovní toky, což jsou aktivity, které probíhají během vývojového procesu. Je rozlišováno celkem devět klíčových pracovních toků, z toho šest hlavních a tři podpůrné. (Shuja 2008)



Obrázek 22: Pracovní toky RUP
Zdroj: Shuja 2008

Perspektiva postupů popisuje optimální postupy, které jsou doporučeny pro vývoj software. Model je založen na šesti základních optimálních postupech (Kadlec 2004)

- *Iterativní vývoj softwaru*, což znamená vývoj softwaru postupně od částí, které jsou pro uživatele nejdůležitější, včetně průběžného získávání zpětné vazby.
- *Správa požadavků*, která by měla sloužit k efektivnímu získávání a uchovávání explicitně zdokumentovaných požadavků uživatelů.
- *Použití architektury založené na komponentách*, tedy rozdělení struktury aplikace na menší jednotky.
- *Vizuální modelování softwaru*, které usnadní srozumitelnost a pochopení zaznamenaných informací, doporučeno je použití modelovacího jazyka UML.
- *Ověření kvality softwaru* určuje, zda aplikace splňuje podnikové standardy kvality a požadavky stanovené na začátku procesu.
- *Řízení změn softwaru,* které řeší, jak postupovat v případě, že je třeba software pozměnit.

Inkrementální vývoj

Inkrementální vývoj spočívá v postupném hledání ideálního produktu pro uživatele. Na začátku je nasazena prvotní verze, která obvykle nemá mnoho funkčností, pouze ty nejdůležitější. Takový produkt bývá označován v oblasti marketingu jako Minimum Viable Product (dále MVP). Jedná se o produkt s nejmenší možnou funkcionalitou, který slouží k otestování reakcí zákazníků. Musí se však jednat o funkční produkt, jinak může dojít k odrazení uživatelů. Postupně jsou sbírány názory uživatelů, které jsou následně vyhodnoceny a podle nich se vyvíjí další verze se zapracovanými požadavky, dokud nevznikne odpovídající software. Inkrementální vývoj softwaru je základem agilního přístupu. Je vhodný v situacích, kdy nejsou předem známy přesné potřeby uživatelů. Nové verze slouží k demonstraci možností uživatelům, kteří následně mohou opět rozhodnout, zda je třeba funkčnost či vzhled poupravit. Aby bylo možné flexibilně reagovat na požadavky uživatelů, je nutné mít dobře zvládnutou organizaci vývojového týmu. Několik verzí softwaru může být vyvíjeno současně, je však třeba předem definovat pořadí jejich nasazení (s tím souvisí i nutnost zavést systém v označování verzí) a vývoj synchronizovat tak, aby do postupně nasazovaných verzí byly zapracovány i žádoucí změny z verzí

předchozích. Při inkrementálním vývoji je vhodné mít k dispozici minimálně dvě oddělená aplikační prostředí – experimentální, kde mohou uživatelé zkoušet a komentovat připravované funkce, a provozní, ve kterém je udržována stabilní verze aplikace maximálně vyhovující požadavkům uživatelů. (Sommerville 2013)

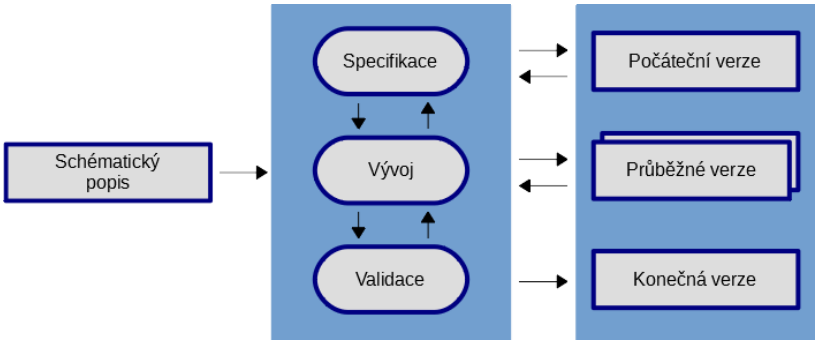
Oproti vodopádovému modelu přináší výhody: (Mujumdar 2012)

- Změny jsou očekávány a uplatňovány pro přizpůsobení aplikace potřebám zákazníka. Náklady na jejich zavedení jsou nižší, související dokumentace k aplikaci se mění dynamicky.
- Stejně tak jsou nižší náklady nasazení prvního prototypu aplikace a zpětnou vazbu od zákazníků je možné získávat mnohem dříve, než pokud by byla vyvíjena rovnou jako kompletní celek. Především v případě cílového trhu s obtížně předpovídatelnými potřebami lze předejít implemetaci zbytečných funkčností a opomenutí funkčností klíčových pro uživatele.
- Prvotní nasazení aplikace se značně urychlí, pokud se vývojový tým omezí jen na klíčové funkce. Především v případě unikátní myšlenky může být tento čas rozhodující z hlediska konkurenčního boje.
- Uživatel si mnohem lépe představí změnu, kterou vidí, než změnu, která je popsána v návrhu, proto může být efektivnější změnu místo dlouhého popisování nějakým způsobem zapracovat a poté s uživateli diskutovat o tom, co by rádi změnili. Díky tomu nemůže dojít k nepochopení návrhu ze strany uživatele. Není třeba hned zpočátku implementovat kompletní funkčnosti. Pokud například uživatel požaduje výběr z více typů platby, můžeme zpočátku vyřešit, jak bude vypadat formulář, nasadit verzi, kde je možné přepínat různé vzhledy, ale prozatím bez možnosti odeslat, na které zatím mohou vývojáři pracovat. Poté se zapracují připomínky týkající se vzhledu a v další verzi je nasazena kompletní funkčnost.
- Průběžné nasazování změn v uživateli vyvolává pocit, že o aplikaci někdo neustále pečuje. Oproti tomu pokud vývojový tým vyvine aplikaci, kterou poté několik let neupraví, může to vzbuzovat dojem, že není aktuální, bezpečná a že vydavatel již neposkytuje pro danou aplikaci podporu, i když mohla být jen navržena tak dobře, že žádné změny nebyly od nasazení potřeba.

Je možné narazit i na několik nevýhod tohoto modelu. Nasazení nehotové aplikace může některé uživatele odradit. Ne každý se o aplikaci zajímá tolik, aby zjistil, že je plánován její rozvoj, a místo aby odeslal své připomínky, přestane aplikaci používat. U nerozhodných uživatelů může být nákladné najít řešení, které jim bude vyhovovat. Také koordinace inkrementálního vývoje především v případě vyvíjení více verzí současně může být náročnější než v případě vodopádového modelu. Vedoucí vývojového týmu může ve vyvíjených verzích snadno ztratit přehled, především pokud není související dokumentace průběžně a včas aktualizována. Další nevýhodou může být nutnost zavedení a spravování nějaké komunikační cesty pro získávání zpětné vazby od uživatelů, což může vést k dodatečným nákladům. Změny v aplikaci, především pokud musí být provedeny narychlo, mohou vést ke snížení kvality kódu. Například pokud vlivem požadovaných změn bude vhodnější pozměnit strukturu aplikace, ale kvůli časovému omezení se zachová stávající struktura na úkor srozumitelnosti či udržovatelnosti kódu. (Mujumdar 2012)

Postup inkrementálního vývoje softwaru je následující: (Sommerville 2013)

- Vývoj první verze je podobný jako v případě vodopádového modelu. Hotová verze je následně nasazena a předána uživatelům ke komentování.
- Ve chvíli, kdy vývojový tým získá zpětnou vazbu od zákazníků, může začít plánovat další verze aplikace.
- Každá nová funkčnost potom prochází zkráceným životním cyklem rovněž podobným jako u vodopádového modelu – specifikací, vývojem a validací. Tyto aktivity se navzájem prolínají a v omezené míře mohou být vykonávány současně.



Obrázek 23: Inkrementální vývoj
Zdroj: Sommerville 2013

2.5.2 Agilní přístup

Na trhu softwarových produktů je vysoká konkurence. Vývojové týmy se musí konkurenčnímu tlaku přizpůsobit. V současnosti již není nejdůležitější bezchybná funkčnost softwaru, ale především spokojenost uživatele, který navíc často mění své potřeby. Je klíčové, aby byl vývojový tým schopný software přizpůsobovat potřebám uživatelů, a to nejlépe v co nejkratším termínu. Tomu se snaží napomáhat agilní metodiky vývoje softwaru.

Od tradičních metodik se liší především v odlišném způsobu pojetí a řízení projektu. V případě agilního vývoje je celkový rozsah požadované funkcionality flexibilní proměnnou, která se upravuje dle potřeb zadavatele či uživatelů, stejně tak lze dle dohody flexibilně měnit také termín dodání i náklady. Samozřejmě ale jen do určité míry, aby byl výsledný produkt přijatelný jak pro zadavatele a sponzora, tak pro cílové uživatele. Oproti tomu kvalita, kterou je především uživatelský zážitek, by měla být klíčovým faktorem, který by se neměl snižovat na úkor termínu ani nákladů. (Kadlec 2004)

Agilní metodiky jsou založeny na manifestu agilního vývoje softwaru. Ten specifikuje základní myšlenky, kterých by se měly agilní metodiky držet. Hlavními myšlenkami jsou tyto (Avison 2009):

- Jednotlivci a interakce mezi nimi jsou důležitější než procesy a nástroje.
- Fungující software je více podstatný než vyčerpávající dokumentace.
- Spolupráce se zákazníkem během vývoje je efektivnější než rozsáhlé vyjednávání o smlouvě před započetím vývoje.
- Schopnost flexibilně reagovat na změny je důležitější než plán samotný.

Pro agilní metodiky platí několik společných principů zobrazených na následujícím obrázku, kterým ale není vždy snadné při vývoji plně vyhovět. (Sommerville 2013)

	Popis
Zapojení uživatele	Do vývojového procesu je třeba zapojit (budoucí) uživatele softwaru. Jejich úkolem je software hodnotit a zadávat nové požadavky na jeho změny a určit jejich prioritu.
Inkrementální dodávání	Nové verze softwaru jsou dodávány inkrementálně na základě průběžně analyzovaných požadavků uživatelů.
Lidé namísto procesu	Organizace vývoje je založena na svobodě vývojářů při volbě postupů. I zde by však měla probíhat určitá kontrola kvality kódu.
Přijímání změn	Je třeba očekávat, že se požadavky na software budou měnit a tomu by měl být přizpůsoben proces vývoje.
Zachování jednoduchosti	S ohledem na minimalizaci nákladů a práce by se měl celý tým snažit zachovat v softwaru i v souvisejících procesech maximální jednoduchost.

Obrázek 24: Principy agilních metodik
Zdroj: Sommerville, 2013

Procesy agilního vývoje softwaru jsou navrženy tak, aby bylo možné rychle dodat softwarový produkt. Software je vyvíjen v řadě inkrementů, které do aplikace přidávají nové funkčnosti. Na začátku je obvykle dodán pouze produkt s minimální možnou funkcionalitou, jehož další rozvoj závisí především na potřebách a názorech uživatelů. (Kadlec 2004)

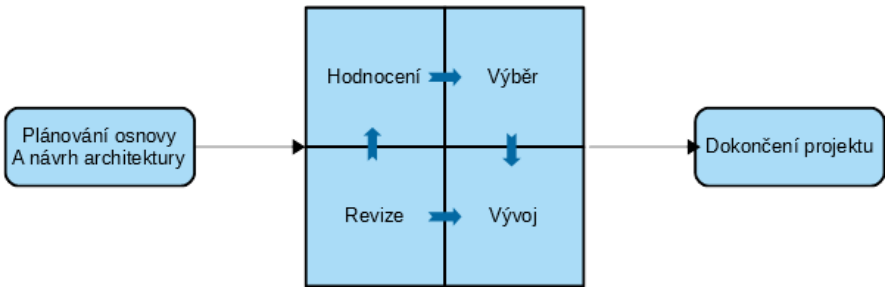
Je však nutné zmínit, že agilní vývoj v některých případech není vhodné použít. Například vývoji bezpečnostních systémů musí bezpodmínečně předcházet důkladná analýza a návrh všech funkcí a vlastností. Klíčové je také zapojení uživatelů ve vývojovém procesu, pokud nebudou ochotni poskytovat zpětnou vazbu, nebude agilní vývoj fungovat. Agilní metodiky jsou naopak vhodné v případě vývoje menších aplikací pro větší množství uživatelů s těžko předvídatelnými potřebami.

Scrum

Scrum je příkladem agilní metodiky vývoje softwaru. Zaměřuje se spíše na způsob řízení vývoje než na samotné techniky používané při vývoji, je vhodné ho tedy doplnit o nějakou metodiku, která je více zaměřena na technické nástroje. Klíčovou myšlenkou této metodiky je opakování rychlých cyklů, nazvaných sprinty, obvykle o délce 2-4 týdny. (Kadlec 2004)

Vývojový proces se skládá ze tří fází. V první fázi dochází k plánování osnovy, která zachycuje hlavní cíle produktu, podle ní je poté navržena architektura aplikace. Poté obvykle následuje řada sprintů, které se skládají ze čtyř opakujících se aktivit: hodnocení, výběr, vývoj a revize. Poslední fází je potom dokončení projektu, kdy je projekt uzavřen a

zhodnocen jeho přínos. Během této fáze může dojít k finálním úpravám dokumentace, uživatelských příruček, nápovědy a dalších souvisejících dokumentů. (Sommerville 2013)



Obrázek 25: Scrum
Zdroj: Sommerville 2013

Cyklus začíná stanovením změn, které je potřeba na softwaru provést. Ve fázi hodnocení jsou k těmto změnám přiřazena rizika. Dále jsou po dohodě se zadavatelem stanoveny priority jednotlivých změn a vybrány změny, které budou součástí verze. Poté může začít samotný vývoj, který je řízen pomocí pravidelných schůzek, na kterých je shrnut dosavadní postup, určeny priority dalšího vývoje a zbývající práce je rozdělena mezi vývojáře. (Sommerville 2013)

Během vývoje už tým obvykle se zadavatelem příliš nekomunikuje. Každý zaměstnanec by měl mít právo provádět rozhodnutí, která se týkají jeho pracovní pozice, vedoucí týmu je pak jen určitým prostředníkem, který na práci dohlíží a komunikuje se zadavatelem. Informace o postupu získává z pravidelných schůzek označovaných pojmem *scrum*. Na konci cyklu je projekt revidován a předložen zadavateli ke kontrole. (Kadlec 2004)

Použití Scrumu při vývoji přináší tyto výhody: (Sommerville 2013)

1. Vývoj je rozdělen na řadu krátkých a kontrolovaných vývojových cyklů.
2. Problémy, na které vývojový tým narazí během vývoje, se ihned řeší na pravidelných schůzkách a neblokují další vývoj.
3. Celý tým má přehled o všem, co se během vývoje děje, zlepšuje se týmová komunikace.
4. Uživatelé vidí pravidelné změny systému, mohou se i podílet na vývoji.
5. Buduje se důvěra mezi vývojovým týmem a zadavatelem/uživateli.

Extrémní programování

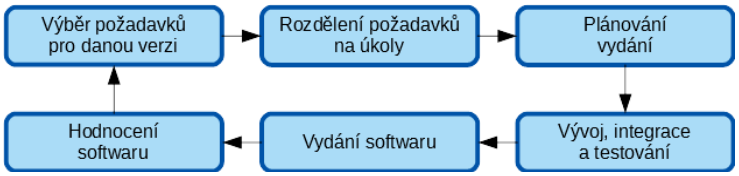
Extrémní programování je jednou z nejznámnějších agilních metodik. Název vznikl ze skutečnosti, že tato metodika většinu agilních přístupů posunuje na extrémní úroveň. Programátoři pracují ve dvojicích a vzájemně si kontrolují kód. Nejprve se píšou testy, potom zdrojový kód komponenty, který na konci vývoje musí všemi testy úspěšně projít. Nové verze jsou vydávány s velmi krátkými časovými rozestupy (obvykle několik dnů či týdnů). Termíny vydání by se neměly zpozdit. Pokud dojde k problémům, po konzultaci se zástupcem uživatelů mohou být některé funkce z další verze odebrány, aby bylo možné termín stihnout. Vývojový tým obvykle z extrémního programování přijme jen několik přístupů, které považuje za nezbytné pro vývojový proces. (Sommerville 2013)

Extrémní programování zahrnuje přístupy vycházející z agilního vývoje (Kadlec 2004):

- Je podporován inkrementální vývoj s malými a častými vydáními systému. Přijímání změn je založeno na pravidelném dodávání nových verzí aplikace uživatelům.
- Získávání zpětné vazby je zajištěno trvalou spoluprací vývojového týmu se zadavatelem či zástupcem uživatelů. Spolupráce je mnohem intenzivnější než u jiných metodik, vývojový tým s touto osobou konzultuje nejen funkčnosti, ale také jejich prioritu, jednotlivé scénáře a další detaily. Je vhodné, aby byl zástupce uživatelů členem vývojového týmu a byl tedy kdykoliv k dispozici.
- Automatické jednotkové testy jsou psány před zahájením programování a slouží vývojářům ke kontrole funkčnosti kódu. Jednotkové testy vycházejí z navržených scénářů.
- Vývoj probíhá ve dvojicích, což zvyšuje kvalitu kódu a relativně rychlý vývoj, ale zachovává myšlenku svobodného vývoje. Dvojicím jsou přidělovány úkoly na úpravu různých částí softwaru, takže nedochází k izolovanosti znalostí. Za napsaný kód potom odpovídá celá dvojice.
- Jednotlivé úkoly jsou ihned po dokončení integrovány do softwarového produktu. Všechny testy musí bez chyby proběhnout před integrací i po ní.
- Přesčasy jsou tolerovány jen krátkodobě ve výjimečných případech, dlouhodobě by totiž mohly vést ke snižování kvality kódu.

- K zachování jednoduchosti se používají jednoduché návrhové vzory a probíhá častý refaktoring pro zlepšení kvality kódu. Při psaní kódu se nebere příliš velký ohled na možnost budoucích změn. Přístup akceptuje změny, ale nesnaží se je předpovídat, místo toho jsou zapracovány až ve chvíli, kdy jsou požadovány zákazníkem. Pokud změny vyžadují zásahy do struktury kódu, je proveden refaktoring.

Vývojový proces má v případě extrémního programování fáze zachycené na následujícím obrázku:



Obrázek 26: Extrémní programování
Zdroj: Mujumdar, 2012

Požadavky uživatelů jsou stanoveny v průběhu tzv. planning game neboli plánovací hry. Název vznikl zčásti ze skutečnosti, že termín hra v lidech vyvolává příjemnější asociace než termín schůzka, a také podle systému karet, který se využívá při plánování změn softwaru. Plánovací hra je nepříliš formální setkání vývojového týmu a zadavatele, kdy je postupně definována série karet, které specifikují požadavky uživatelů. Tato aktivita se skládá ze tří fází: (Kadlec, 2004)

1. První fází je *průzkum* (exploration), kdy jsou sepsány tzv. user stories, tedy uživatelské příběhy, které popisují určitou požadovanou funkci softwaru. Vývojový tým odhadne náročnost implementace.
2. Ve druhé fázi označované jako *závazek* (commitment) se jedná o rozsahu změn, které budou součástí další verze aplikace. Uživatelské příběhy jsou tříděny a je z nich následně vybráno několik, které budou realizovány. K nim se následně sepíší karty zadání, které obsahují jednotlivé úkoly související se zapracováním změny a jejich hlavní a alternativní scénáře. Je také stanoven termín vydání další verze.
3. Třetí fází je *řízení* (steering) vývoje. Jednotlivé úkoly jsou rozděleny mezi dvojice vývojářů, kteří nejprve připraví testy a poté píšou samotný kód, který je po napsání otestován pomocí připravených testů a integrován do softwarového produktu.

2.6 Softwarová architektura

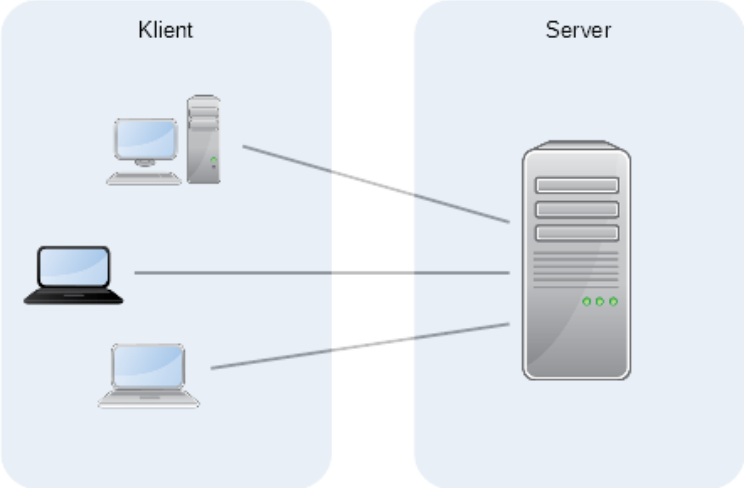
Návrh softwarové architektury slouží k popisu struktury aplikace. Identifikuje hlavní komponenty softwaru a vztahy mezi nimi. Výsledkem procesu návrhu architektury je model architektury, který popisuje, jakým způsobem je software organizován do sady komponent. Architektura softwaru má vliv na výkon, robustnost, distribuovatelnost a možnosti údržby systému.

Softwarová architektura se liší pro centralizované systémy, tedy systémy běžící na jednom stroji, a pro distribuované systémy, které jsou sestaveny z několika nezávislých zařízení, uživateli se však jeví jako jeden koherentní systém.

V současnosti se často využívají vzory, které umožňují prezentovat, sdílet a opakovaně používat znalosti softwarových systémů. Architektonický vzor je možné definovat jako abstraktní popis optimálního postupu, který je ověřen opakovaným použitím pro různé typy systémů. (Sommerville 2013) V následujících podkapitolách bude popsáno několik architektonických vzorů, které mají za cíl usnadnit návrh aplikace.

2.6.1 Architektura Klient-server

Tento návrhový vzor je předchůdcem vrstvené architektury a aplikaci rozděluje do dvou částí – klientské a serverové. Server poskytuje klientské části služby a ta zajišťuje interakci s uživatelem. Klientská část je obvykle aplikace běžící na koncovém zařízení uživatele či v internetovém prohlížeči. Server přijímá obvykle požadavky od mnoha klientů, a proto musí být zaveden systém postupného vyřizování těchto požadavků, aby nedošlo k jejich ztrátě v případě vyššího vytížení serveru. (Gála 2009) Na následujícím obrázku je zachyceno schématické znázornění této architektury.



Obrázek 27: Klient-server
Zdroj: TechDifferences 2017

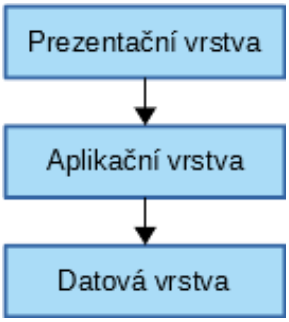
2.6.2 Vrstvená architektura

Tento vzor umožňuje navrhovat strukturu programů, které lze rozdělit na skupiny dílčích úkolů. Třívrstvá architektura definuje tři úrovně abstrakce, každá úroveň poskytuje služby pro následující úroveň. Existují i víceúrovňové architektury, ale na ty se tato práce zaměřovat nebude.

Nejnižší úrovní třívrstvé architektury je *datová vrstva*. Touto vrstvou je obvykle databáze a rozhraní pro její správu. Má za úkol data uchovávat, přidávat, upravovat či mazat, zajišťovat jejich integritu, správnost a konzistenci. (Patel 2014)

Datová vrstva poskytuje své rozhraní *aplikační vrstvě*, která má na starost fungování aplikace. Obsahuje aplikační logiku, tedy všechny případy užití, které lze spustit, jsou součástí této vrstvy. Zde se také zpracovávají data pro uložení do datové vrstvy či pro zobrazení uživateli. (Patel 2014)

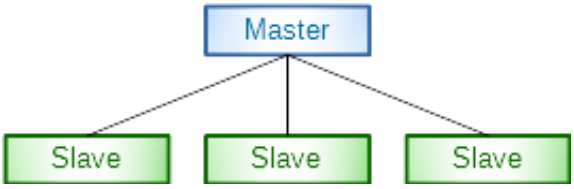
Poslední vrstvou je vrstva *prezenční*, která zajišťuje interakci s uživatelem. Získaná data z aplikační vrstvy převádí do vizuální podoby. (Patel 2014)



Obrázek 28: Třívrstvá architektura
Zdroj: Mallawaarachchi 2017

2.6.3 Architektura Master-slave

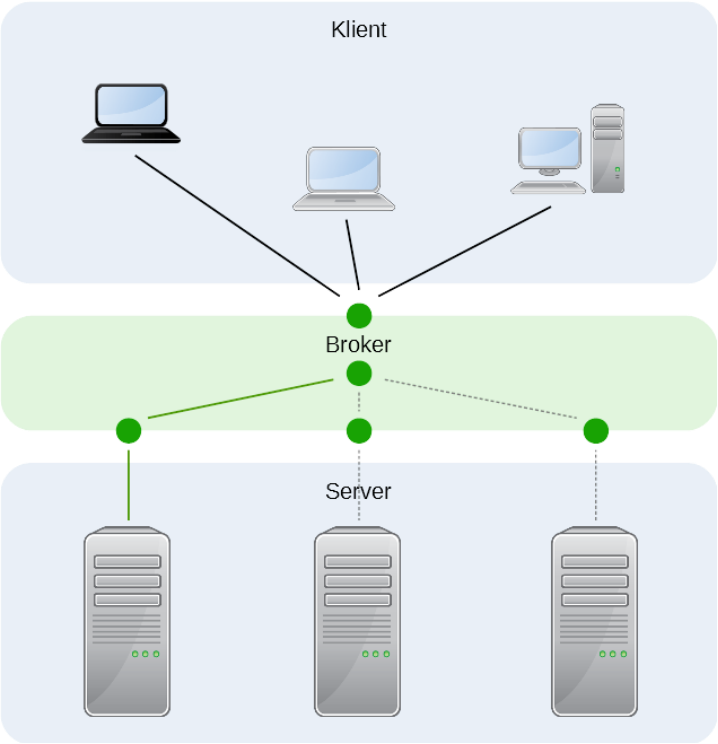
Architektonický vzor označený jako master-slave je architektura pro distribuované systémy, lze jej tedy využít v případě, že je možné aplikaci rozdělit na několik výpočetních jednotek. Předpokládá se, že jedna z výpočetních jednotek bude řídící (master) a bude mít na starosti koordinaci činností ostatních výpočetních jednotek (slaves), kterými se obvykle řeší jeden konkrétní problém či několik souvisejících. (Sommerville 2013)



Obrázek 29: Master-slave
Zdroj: Mallawaarachchi 2017

2.6.4 Architektura Broker

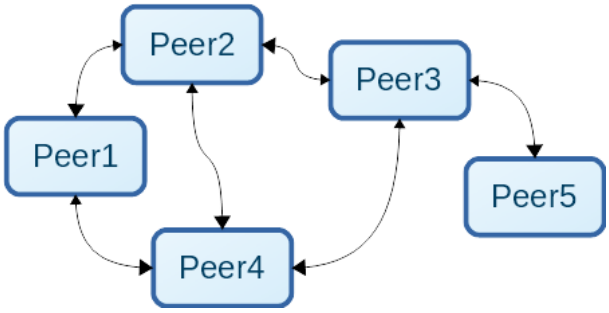
Architektonický vzor Broker je založen na prostředníkovi, který komunikuje s klienty a koordinuje rozesílání zpráv mezi jednotlivými serverovými částmi. Používá se na strukturování distribuovaných systémů, které obsahují oddělené komponenty, které komunikují prostřednictvím vzdáleného volání služeb. (Lewin 1998)



Obrázek 30: Architektura broker
Zdroj: Mallawaarachchi 2017

2.6.5 Architektura Peer-to-peer

Architektonický vzor Peer-to-peer je založen na komunikaci nezávislých komponent, tzv. *peerů*. Jedná se o decentralizovaný systém, ve kterém může provádět výpočty libovolný síťový uzel. Celkový systém je navržen tak, aby využil výpočetního výkonu rozsáhlé sítě počítačů. Tento architektonický vzor nerozlišuje klienty a servery. (TechDifferences 2017)

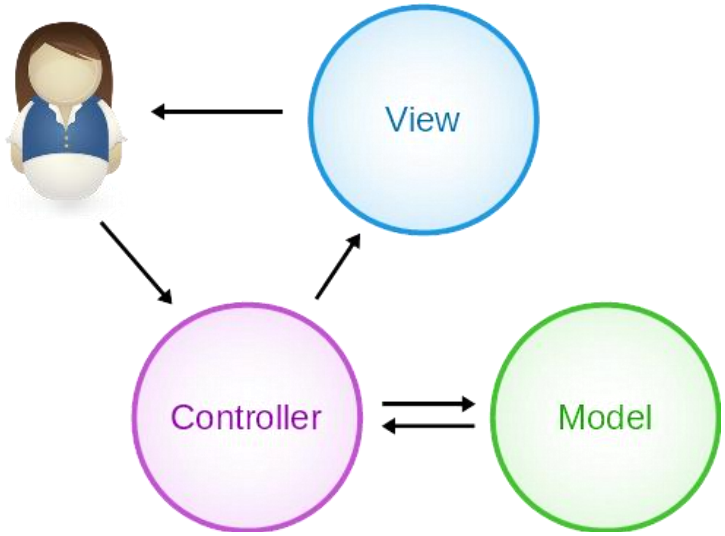


Obrázek 31: Peer-to-peer
Zdroj: TechDifferences 2017

2.6.6 Architektura Model-view-controller

Model-view-controller je architektonický vzor, který odděluje prezentaci a interakci od uživatelských dat. Systém je rozdělen do tří komponent, které vzájemně komunikují: (Sommerville 2013)

- Komponenta označovaná jako *model* spravuje systémová data a provádí s nimi přidružené operace.
- Komponenta *view* má na starosti prezentaci dat uživateli.
- Komponenta *controller* řídí interakci s uživatelem.



Obrázek 32: Model-view-controller
Zdroj: Mallawaarachchi, 2017

2.6.7 Bezserverová architektura

Bezserverová (serverless) architektura je novinka, která se těší čím dál větší oblibě. Při jejím použití se vývojový tým nemusí soustředit na správu serveru a výpočetní infrastruktury, ale na programování funkčností, které přidávají hodnotu aplikaci. To je hlavní myšlenka této architektury.

Serverless architekturu nabízejí hlavní poskytovatelé cloudových služeb, například Microsoft, Amazon, či IBM. Vývojový tým si může zvolit, které části serveru budou

poskytovány prostřednictvím cloudu. Může například využít jen služby pro správu databáze či speciální API, které umožňuje budovat kompletní bezserverové aplikace.

Tato architektura má tři hlavní výhody. Server spravuje poskytovatel cloudu a ten také zajišťuje jeho dostupnost. Architektura se škáluje, server několik hodin nemusí běžet vůbec, když to není potřeba, ale jakmile má zpracovat nějaké požadavky, okamžitě poskytne potřebný výpočetní výkon i na zpracování desítek tisíc souběžných požadavků. Platí se jen za prostředky, které jsou skutečně využity. (Roberts 2016)

Bezserverová architektura bývá také označována jako další typ cloudové služby – Functions as a Service, zkráceně FaaS. (Roberts 2016)

3. Analýza požadavků

Tato část se zaměřuje především na hodnotu software pro cílové uživatele. Konkurence ve světě aplikací je vysoká, proto je třeba vyvíjet takové aplikace, které budou uživatelsky co nejpřívětivější. Každý z uživatelů má však trochu jiné potřeby a vyhovět všem zkrátka není možné. Změna, která jednoho uživatele potěší, může jinému překážet.

3.1 Softwarové požadavky

Pro pojem požadavek existuje mnoho odlišných definic a jedná se tedy o velice abstraktní pojem, který je pro potřeby této práce třeba více upřesnit. Je jasné, že uživatel si pod pojmem požadavek představí něco úplně jiného než vývojář. Pro uživatele je požadavkem například představa, že by rád mohl aplikací vytvořený obrázek rovnou odeslat prostřednictvím e-mailu. Něco takového však vývojáři nemůže stačit k tomu, aby požadavek do aplikace zapracoval, ten potřebuje mnohem konkrétnější zadání. Vzhledem k tomu, že neexistuje jednotné pojetí požadavků, je vhodné, aby si každý vývojový tým pojmy sjednotil. Stejně tak budou pojmy sjednoceny v této práci. V zásadě existují dva základní pohledy na požadavek, které je vhodné během vývoje odlišeny dvěma odlišnými pojmy. Zvolené pojmy by ale měly být jednotně používány všemi, kdo se na vývoji podílí. (Wiegers 2008)

- První pohled na požadavek je obecnější, nejsou přesně stanovena pravidla na jeho podobu. Jedná se o abstraktní pojem vyjadřující potřebu určitého chování či vlastností softwarového produktu. Pro potřeby této práce bude pro toto pojetí používán pojem *softwarový požadavek*.
- Pokud chce vývojový tým uživateli vyjít vstříc a jeho myšlenky zapracovat, je potřeba je nějakým způsobem zaznamenat. K tomu slouží standardizovaný dokument, v němž je potřebné chování jasně formulováno tak, aby na jeho základě bylo možné implementovat daný software. Musí nést určité náležitosti, které si vývojový tým stanoví. Pro potřeby této práce se tento pohled bude označovat jako *specifikace požadavku*.

Někdy se místo požadavku a jeho specifikace používají pojmy uživatelský požadavek (abstraktní potřeba nějaké funkce) a systémový požadavek (ta samá potřeba sepsaná ve formě, kterou požaduje vývojář, aby mohl funkci naprogramovat). V souvislosti

s požadavky se používají i další přídavná jména a jejich použití je vždy na konkrétním vývojovém týmu. (Wiegers 2008)

V dalších podkapitolách bude nejprve konkrétněji popsána specifikace požadavku, proces inženýringu požadavků a zohledněny dva druhy členění požadavků: podle účelu a podle vztahu k funkčnostem.

3.1.1 Specifikace požadavků

Specifikace požadavku musí být (Roudenský 2016):

- Úplná (musí specifikovat veškeré podrobnosti potřebné k implementaci)
- Konzistentní (ve specifikaci požadavku nesmí existovat rozpory, a to jak vzájemně mezi jednotlivými částmi požadavku, tak ve vztahu k software či interním postupům a dalším omezením)
- Sledovatelná (musí existovat možnost, jak sledovat, jaký vliv bude mít požadovaná změna na ostatní části software)
- Proveditelná (požadovanou změnu musí být možné implementovat, je tedy potřeba důkladně analyzovat předpoklady pro zapracování požadavku)
- Udržovatelná (zapracování požadované změny nesmí vést k zásadnímu snížení možnosti údržby software)
- Testovatelná (po implementaci musí být možné otestovat, že zapracované změny odpovídají specifikaci požadavku)

Každou specifikaci požadavku je nutné posuzovat z hlediska možných rizik, bezpečnosti, SW a HW omezení a dalších aspektů, které by mohly být požadovanou změnou ovlivněny. (Roudenský 2016)

Specifikaci požadavků je možné zapisovat ve dvou úrovních podrobnosti. První úroveň slouží uživateli či zadavateli. Je vhodné po sjednání změn softwaru zapsat, na čem se zúčastněné strany dohodly. Pro tuto úroveň není potřeba příliš technických informací ani podrobný rozpis kroků. Druhou úrovní je podrobný rozpis funkcí, podle kterých budou vývojáři psát kód. Zde je naopak žádoucí uvádět co nejvíce podrobností, alternativní scénáře, doporučené technologie a cokoliv dalšího, co by vývojáři mohlo být v průběhu

vývoje nejasné. Také z důvodu následné akceptace je důležité, aby se zúčastněné strany dohodly na co nejkonkrétnějším a co nejpodrobnějším zadání požadavku. (Roudenský 2016) Podle formy zápisu se specifikace požadavků člení na strukturovanou a na specifikaci v přirozeném jazyce. (Sommerville 2013)

- *Strukturovaná specifikace požadavků* vychází z postupů a zkušeností vývojového týmu. Je třeba stanovit strukturu specifikace požadavku, tedy informace, které je ke každému požadavku potřeba zaznamenat. Jedná se především o vstupy a výstupy, hlavní scénář a alternativní scénáře, podmínky pro spuštění a stav po doběhnutí.
- *Specifikace požadavků v přirozeném jazyce* dokáže předat variabilní množinu informací, je vhodné ji použít v případě, že vývojový tým předem netuší, jaké informace je potřeba uchovávat. Především s možností použití obrázků je vhodným nástrojem, pokud je specifikace požadavků kreativní proces, jako je tomu u některých metodik. Pro vývojáře je ale obvykle náročnější se v nestrukturovaném textu orientovat a vyhledávat v něm potřebné informace.

3.1.2 Inženýrství požadavků

Proces, během kterého jsou požadavky zjišťovány a zapisovány, se nazývá *inženýrství požadavků*. Inženýring požadavků provádí vývojový tým, ale klíčová je také spolupráce se zadavatelem a uživateli. Tento proces je založen na dvou hlavních dokumentech. Prvním z nich je *studie proveditelnosti*, která slouží pro ověření, zda je možné a vhodné daný software (či požadované změny) vyvíjet. Druhým důležitým dokumentem je již zmíněná *specifikace požadavků*, o které pojednává následující podkapitola. Tento proces lze rozdělit do čtyř základních fází: (Sommerville 2013)

1. *Zjišťování požadavků* spočívá v hledání informací o vyvíjeném softwaru a o jiných podobných aplikacích. V této fázi se komunikuje se zainteresovanými osobami a hledají se jejich potřeby.
2. *Analýza požadavků* slouží k bližšímu prozkoumání požadavků jednotlivých osob, které obnáší především hodnocení efektivnosti jejich zapracování do aplikace. Součástí této fáze je také studie proveditelnosti.
3. *Specifikace požadavků* je fáze, během které jsou požadavky sepsány do standardizovaného dokumentu tak, aby bylo možné podle něj požadované funkce do

softwaru zapracovat. Výstupem této fáze je potom dokument se specifikovanými požadavky.

4. *Validace požadavků* spočívá v ověřování, zda požadavky skutečně definují software, který zadavatel požaduje a který uživatelé budou ochotni používat. Zaměřuje se na hledání problémů a chyb v požadavcích.

3.1.3 Členění podle účelu

Obecně lze softwarové požadavky rozdělit do čtyř základních skupin: podnikatelské, uživatelské, funkční a parametrické. (Wiegers 2008)

- *Podnikatelské požadavky* formulují podnikatelské cíle, kterých chce zadavatel dosáhnout prostřednictvím využití daného softwaru.
- *Uživatelské požadavky* popisují cíle uživatelů a jejich potřeby.
- *Funkční požadavky* definují, jakými funkcemi (neboli features) má software disponovat.
- Potom lze odlišit *požadavky parametrické*, které ale nejsou požadavky osob, spíše se jedná o určitá omezení, jak v souvislosti s používaným softwarem a hardwarem, tak s interními předpisy, legislativou atd. Parametrické požadavky nevyplývají z funkce softwaru, ale z prostředí, ve kterém je provozován.

3.1.4 Členění podle vztahu k funkčnostem

Požadavky lze rozdělit na funkční a mimofunkční. (Sommerville 2013)

- *Funkční požadavky* popisují vlastnosti a chování softwaru. Mohou také explicitně definovat, jak se aplikace za žádných okolností chovat nesmí.
- *Mimofunkční požadavky* zahrnují různá omezení funkcí či vlastností, které systém poskytuje. Příkladem mimofunkčních požadavků může být výkon, bezpečnost či dostupnost. Mimofunkční požadavky jsou často důležitější než funkční požadavky, pokud při vývoji nejsou respektovány, může to vést k nefunkčnosti celé aplikace. Mimofunkční požadavky se dále dělí do tří skupin:

- *Produktové požadavky* omezují chování systému. Mohou souviset s jeho výkonem, dobou odezvy a dalšími důležitými vlastnostmi.
- *Organizační požadavky* se odvíjejí od postupů používaných vývojovým týmem a dalšími klíčovými rolemi.
- *Externí požadavky* jsou způsobeny vnějšími faktory, které vývojový tým nedokáže ovlivnit, jako je například legislativa.

3.2 Kvalita SW

Kvalita je především dnes pro spoustu osob hlavním kritériem, podle kterého vybírají produkty. U softwaru tomu není jinak. Vývoj kvalitního softwaru znamená pro vývojový tým konkurenční výhodu. Hlavním problémem v této oblasti je, že kvalitu nelze snadno změřit a pro různé osoby může být vnímání kvality stejného produktu zcela odlišné. Jak tedy vyrábět kvalitní software? Jak vůbec zjistit, co cílový uživatel bude považovat za kvalitní software?

Co si představit pod pojmem kvalita, mohou pomoci objasnit následující definice (Roudenský 2016). Podle Philipa Crosbyho je kvalita „soulad s definovanými požadavky“. Dr. Joseph Juran popsal kvalitu jako „vhodnost pro zamýšlené používání“ a definoval 3 základní procesy:

1. Plánování kvality, které spočívá v identifikaci zákazníků a jejich potřeb a následném návrhu produktu, který dokáže potřeby spotřebitele uspokojit.
2. Řízení kvality, které hodnotí současný produkt.
3. Zlepšování kvality, tedy hledání možností, jak napravit nedostatky a produkt nadále rozvíjet.

Podle Davida Garvina lze na kvalitu pohlížet z pěti odlišných pohledů:

1. Transcendentální pohled – kvalita představuje určitou formu ideálu, kterou je možné rozeznat na základě zkušeností a intuice.
2. Pohled orientovaný na uživatele – kvalita je mírou uspokojení potřeb uživatelů.
3. Pohled orientovaný na výrobu – kvalita je mírou dodržení specifikovaných požadavků.
4. Pohled orientovaný na produkt – kvalita je určena hodnotami určitých měřitelných atributů a lze ji objektivně posoudit.

5. Pohled orientovaný na hodnotu – kvalita znamená schopnost vyhovět požadavkům za přijatelnou cenu.

Kritérií pro posouzení kvality je mnoho a často jsou velmi subjektivní a závislá na dalších aspektech, především na rozpočtových, časových či jiných omezeních, která ovlivňují rozhodování spotřebitele. Některé lze v omezené míře kvantifikovat, jejich hodnocení či důležitost však je možné pouze odhadovat. Přesto existují doporučené postupy, které mohou pomoci pochopit, co je kvalitní software a jak při vývoji kvality dosáhnout. Kvalitu je potřeba potřeba hodnotit u jednotlivých procesů i u jejich výstupů. Do produktu je kvalita vestavována v průběhu celého životního cyklu vývoje. V této souvislosti se často mluví o tzv. nákladech na kvalitu, které mají zohlednit finanční prostředky vynaložené na nalezení a opravu nedostatků. Při posuzování kvality software je vhodné použít modely kvality softwaru, o kterých pojednává následující podkapitola. (Roudenský 2016)

Existuje mnoho atributů softwaru, které s kvalitou souvisí, například bezpečnost, srozumitelnost, přenositelnost, spolehlivost, zabezpečení, testovatelnost, odolnost, složitost, efektivita, použitelnost a další. Je subjektivní, v jaké míře dané atributy ovlivňují u uživatelů vnímání kvality softwaru. (Sommerville 2013) Některé z těchto vlastností budou popsány v následujících kapitolách.

3.2.1 Modely kvality SW

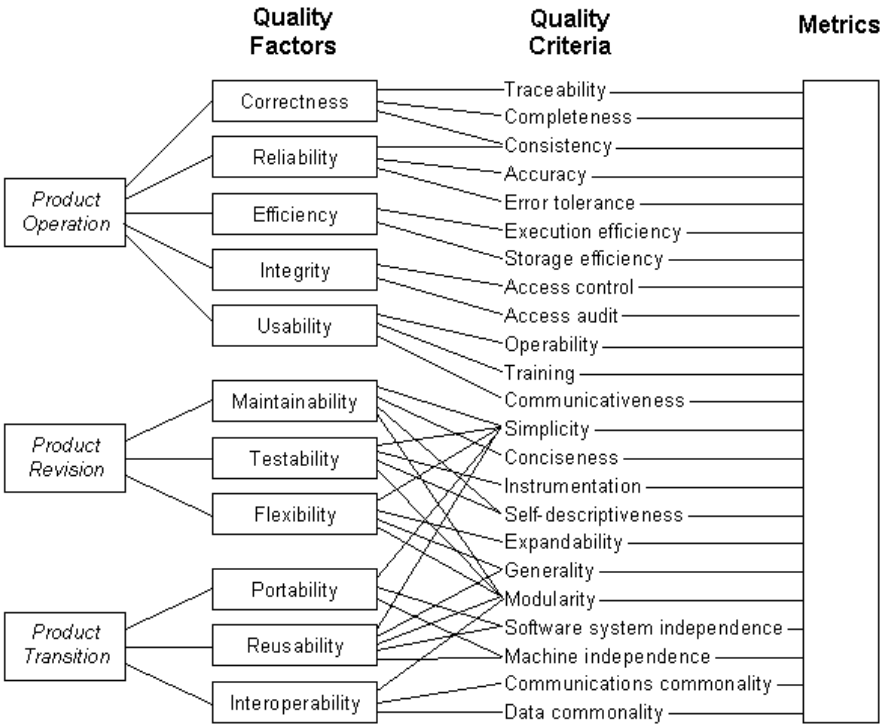
Vzhledem k tomu, že je kvalita těžko uchopitelný pojem, existují různé modely, které kvalitu zkoumají z odlišných pohledů. Lze je rozdělit do dvou základních skupin (Roudenský 2016):

1. Procesní modely kvality softwaru se zaměřují na procesy, které s vývojem souvisí. Vychází z předpokladu, že dobře naplánované a dobře řízené procesy vedou ke kvalitnímu produktu. K měření kvality využívají procesní míry (například délka trvání činnosti, nákladová efektivnost).
2. Produktové modely kvality softwaru se zabývají přímo produktem a k tomu využívají produktové míry (například udržitelnost, množství funkcí).

McCallův model

Jedním z prvních modelů kvality byl model představený McCallem v roce 1977. Snaží se eliminovat rozdíly mezi vývojáři a uživateli a zaměřuje se především na takové faktory, které jsou důležité pro obě strany, jako je například spolehlivost či rozšiřitelnost. Na

obrázku níže je možné vidět, jaké tři pohledy na kvalitu McCall definoval a jaké faktory a kritéria k nim přísluší. (Roudenský 2016)



Obrázek 33: McCallův model
Zdroj: Burris, 2007

FURPS

Dalším příkladem modelu kvality je model FURPS z roku 1987. Jeho název vznikl jako zkratka z prvních písmen označení faktorů kvality, kterými se zabývá, tedy (Al-Badareen 2011):

- Functionality (funkčnost)
- Usability (použitelnost)
- Reliability (spolehlivost)
- Performance (výkon)
- Supportability (udržovatelnost)

Dnes se používá především rozšířená podoba tohoto modelu FURPS+, která kromě předchozích charakteristik zahrnuje i několik dalších kritérií týkajících se návrhu či vývoje SW: (Roudenský 2016)

- Design constraints (omezení návrhu v podobě závislosti na softwarových či hardwarových platformách)
- Implementation requirements (způsob vývoje včetně zvolených technologií)

- Interface constraints (způsob komunikace s jinými systémy či aplikacemi)
- Physical requirements (charakteristiky HW součástí systému)

Normalizace

V oblasti kvality software je důležité zmínit řadu norem ISO/IEC 9126 *Jakost softwarového produktu*. Model kvality dle normy ISO/IEC 9126 má hierarchickou strukturu. Základem normy je definice šesti charakteristik jakosti software a čtyř charakteristik jakosti při používání, které se dále rozpadají na příslušné podcharakteristiky a ty jsou určeny jedním či více přímo měřitelnými atributy: (Roudenský 2016)

- Charakteristiky jakosti softwarového produktu
    - Funkčnost
    - Bezporuchovost
    - Použitelnost
    - Účinnost
    - Udržovatelnost
    - Přenositelnost
- Charakteristiky jakosti při používání softwarového produktu
    - Efektivnost
    - Produktivita
    - Bezpečnost
    - Uspokojení

Atributy pro posouzení kvality softwarového produktu jsou rozděleny na vnitřní, vnější a odděleně je zkoumána jakost při používání. Předpokládá se, že vnitřní atributy ovlivňují vnější atributy kvality a ty zase ovlivňují jakost při užívání. S touto normou úzce souvisí i řada norem ISO/IEC 14598 *Hodnocení softwarového produktu*, která byla určena pro hodnocení softwarového produktu během vývoje, a norma ISO/IEC 12119 *Informační technologie – Softwarové balíky – Požadavky na jakost a zkoušení*, která byla zaměřena na hodnocení produktu ze strany uživatele. (Roudenský 2016)

Existovala ještě celá řada dalších norem, které se kvality software týkaly, přičemž vznikaly odděleně bez vzájemné provázanosti, byly nekonzistentní, nekompletní, a v jejich číslování nebyl zaveden systém. V roce 1999 byl zahájen projekt SQuaRE (Software Quality Requirements and Evaluation – požadavky na kvalitu software a hodnocení), který měl za úkol tyto normy sjednotit a poskytnout logicky organizované a jednotné řady norem pro hodnocení kvality software. (Roudenský 2016)

Projekt SQuaRE vedl ke vzniku rodiny vzájemně provázaných norem a technických zpráv, pro něž je vyhrazeno číselné označení ISO/IEC 25000 – 25099, přičemž prozatím jsou využita označení 2500x – 2504x, zbytek je vyhrazen jako rezerva pro normy, které vzniknou v budoucnu. Normy a technické zprávy jsou členěny do pěti oddílů (ISO/IEC 25000:2014):

- 25000x: Quality Management Divison (oddíl řízení kvality)
- 25010x: Quality Model Divison (oddíl modelu kvality)
- 25020x: Quality Measurement Divison (oddíl měření kvality)
- 25030x: Quality Requirements Divison (oddíl požadavků na kvalitu)
- 25040x: Quality Evaluation Divison (oddíl hodnocení kvality)

3.2.2 Management kvality SW

Management kvality je snaha o neustálé zlepšování procesů ve firmě, jehož cílem je vyšší efektivita a produktivita a nižší náklady na vývoj SW. Nejčastěji je realizován prostřednictvím čtyř základních nástrojů: (Roudenský 2016)

- Plánování kvality spočívá v definování krátkodobých a dlouhodobých cílů v oblasti kvality a vytvoření plánu k uskutečnění vytyčených cílů.
- Řízení kvality zahrnuje takové činnosti, které slouží ke kontrole, zda je plán dodržen a zda se dosahuje požadované kvality jak u procesů, které s vývojem SW souvisí, tak u dílčích a výsledných produktů, a také činnosti, které mají za úkol napravit nedostatky nalezené během kontroly.
- Prokazování kvality může být chápáno jako souhrn prostředků, které slouží k prokázání schopnosti organizace vyhovět požadavkům na kvalitu vyvíjeného SW.
- Zlepšování kvality slouží k neustálému zlepšování procesů a tím i ke zlepšování kvality finálního produktu.

3.2.3 Kvalita kódu

Kvalita softwaru jde ruku v ruce s kvalitou jeho zdrojového kódu. Proto je nezbytné zavést standardy, které budou vývojáři dodržovat, kód pravidelně revidovat především po nových

členech týmu, a provádět další podpůrné činnosti, které jeho kvalitu mohou kontrolovat a zvyšovat.

Kvalitní kód lze identifikovat podle následujících znaků:

- Kód musí být věcně správný, definované funkčnosti jsou v souladu s návrhem a potřebami zadavatele a koncových uživatelů. Je důležité, aby v software oproti návrhu nic nechybělo, ale také aby neobsahoval nevyžádané funkčnosti. Je nežádoucí, aby si vývojáři přidávali funkce například pro usnadnění dohledávání chyb, aniž by byly tyto změny schválené a zapsané v návrhu. Nejenže to snižuje udržovatelnost kódu, ale může to přinést i závažná bezpečnostní rizika.
- Kód by měl být psán efektivně. To je jeden z aspektů kvality kódu, který se kontroluje obtížně. Efektivní kód využívá nejvhodnější postupy pro dosažení konkrétního cíle. Příkladem neefektivního postupu může být neukončení cyklu pro prohledávání pole po nalezení hledaného prvku či použití nevhodných knihoven pro realizaci určité funkčnosti.
- Musí být dobře udržovatelný, to znamená, že je snadné přidávat nové funkčnosti i modifikovat či odstraňovat již existující. Kód by měl být přehledně strukturovaný, okomentovaný, je důležité dodržovat stanovené konvence, třídy, metody, proměnné a další komponenty kódu pojmenovávat jedním jazykem (ideálně anglicky).
- Udržovatelnost jde ruku v ruce s dostupností kódu. Vývojový tým si musí stanovit jasná pravidla pro sdílení zdrojového kódu uvnitř týmu. Je nemyslitelné, aby se práce na vývoji zastavila jen proto, že vývojář onemocní a rozepsaný zdrojový kód si s sebou odnese domů.
- Zdrojový kód by měl být pokryt jednotkovými testy v celém rozsahu, což snižuje náchylnost na chyby. Je však důležité, aby i jednotkové testy byly kvalitní, jinak nemá jejich použití smysl. Jednotkové testy mohou také pomoci při zapracování drobných úprav pro rychlé otestování všech základních i alternativních scénářů.
- Zdrojový kód by měl být vždy psán podle standardů zavedených ve vývojovém týmu. Obvykle si každý tým stanoví alespoň základní konvence pro psaní zdrojového kódu. Ty se dělí především do tří hlavních skupin (Roudenský 2016):

- Formátovací konvence, které usnadní čitelnost kódu a jeho sdílení uvnitř vývojového týmu. Obvykle se jedná o formátovací pravidla nastavená ve vývojovém prostředí, které následně kód formátuje automaticky.
- Jmenné konvence, které zajišťují jednotnost pojmenování jednotlivých elementů programovacího jazyka, čímž se zvyšuje jeho čitelnost a srozumitelnost.
- Konvence pro komentování kódu, které stanovují, k čemu je třeba psát komentáře a v jakém formátu, často se následně využívají nástroje, které ze zdrojového kódu a komentářů vytvoří dokumentaci.

Existují různé nástroje, které slouží ke kontrole dodržování standardů, formátování kódu, měření pokrytí zdrojového kódu jednotkovými testy.

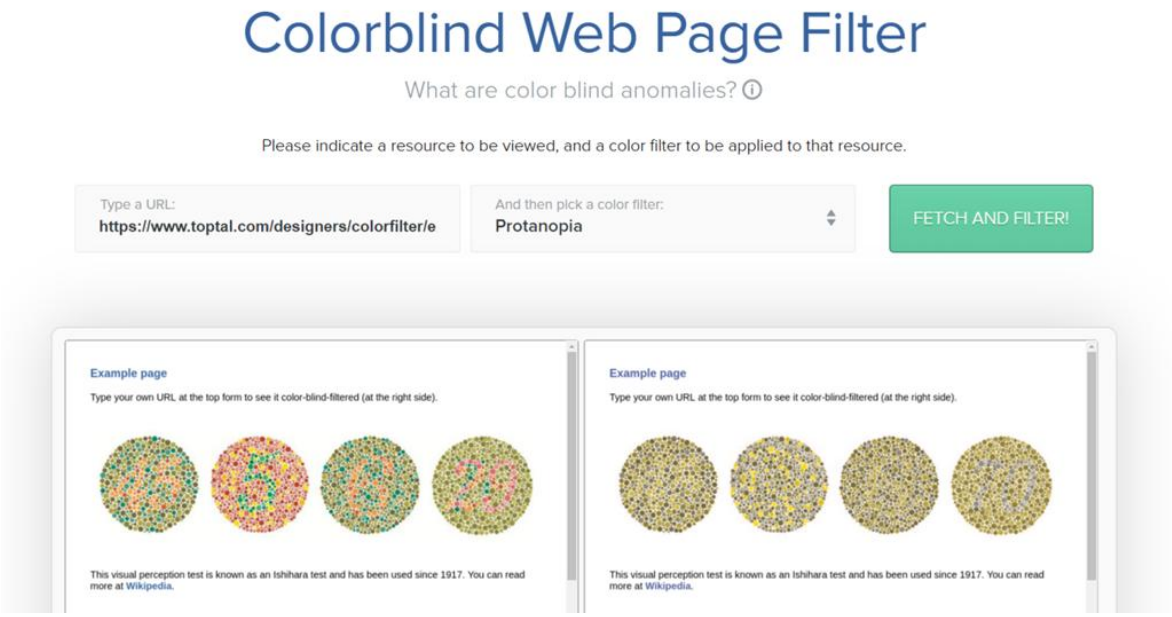
3.3 User Experience

Jako označení pro míru uspokojení potřeb uživatele užíváním určité aplikace se v současnosti často využívá pojem User Experience (zkráceně UX), což se do češtiny překládá jako uživatelský prožitek či požitek, případně zážitek. Týká se především uživatelského rozhraní, tedy té části aplikace, která slouží k interakci s uživatelem. Souvisí ale také s dostupností aplikace a v podstatě se vším, co může ovlivnit spokojenost uživatele při užívání aplikace.

Různé cílové skupiny uživatelů budou mít rozdílné potřeby, proto každému uživateli aplikace přinese jinou úroveň uživatelského prožitku. Existují ale i obecné principy, kterými je vhodné se při vývoji řídit. Další informace o potřebách konkrétní skupiny uživatelů se pak získávají prostřednictvím marketingových průzkumů a analýzy požadavků.

User Experience Design (zkráceně UxD) je sadou technik a metod, které lze použít pro návrh uživatelského rozhraní. Největší smysl má v současnosti na mobilních zařízeních, které přinášejí nové možnosti interakce. Zahrnuje několik dílčích disciplín: (Asociace UX 2018)

- *Uživatelský výzkum* zahrnuje rozhovory, on-line výzkum a další prostředky, které mají za cíl definovat potřeby uživatelů. Bez komunikace s uživatelem nemůže dobrý UX design vzniknout. Nikdo nedokáže předpovědět potřeby uživatelů tak dobře, jako oni sami. V této oblasti je možné využít různé hardwarové nástroje, jako je například kamera pro snímání pohledu. A to jak pro prvotní průzkum, tak pro následné testování.
- *Interakční design* (Interaction Design, zkráceně IxD) se zabývá návrhem interakce s uživatelem pro konkrétní aplikaci. Na mobilních zařízeních se nejedná o pouhá kliknutí, ale o různé typy gest, od posunu až po malování obrázků, které aplikace vyhodnotí. V současnosti je snaha o rozmístění možností interakce tak, aby je uživatel snadno zvládl jednou rukou, ideálně palcem.
- *Informační architektura* (Information Architecture, zkráceně IA) definuje strukturu celé aplikace. Zahrnuje rozdělení obsahu aplikace na jednotlivé sekce, vzájemnou navigaci mezi nimi atd. Cílem je zajistit, aby se uživatel vždy dostal k informaci, kterou potřebuje, a to co nejkratší cestou, tedy co nejnižším počtem kliknutí.
- *Vizuální návrh* potom slouží k definování vzhledu aplikace. Jsou respektovány různé zásady pro vzbuzování emocí pomocí kombinací barev, rozmísťování prvků tak, aby uživatel nemusel provádět zbytečná kliknutí či pohyby myší navíc. Při návrhu vzhledu je třeba brát ohledy i na osoby s poruchou zraku. Ať už se jedná o funkci aplikace, která umožní text zvětšit, či online využití nástrojů, které simulují, jak vidí danou stránku člověk s různými poruchami barvocitu. Na následujícím obrázku je zachyceno použití nástroje Colorblind Web Page Filter, který slouží k tomuto účelu.



Obrázek 34: Colorblind Web Page Filter
Zdroj: Toptal 2017

Uživatelský zážitek úzce souvisí s použitelností produktu, ale zahrnuje také další aspekty, například přehlednost či bezpečnost aplikace. Uživatelský zážitek je možné definovat jako vnímání a reakce jednotlivce plynoucí z užívání nebo předpokládaného využití produktu. V současnosti je na trhu softwaru obrovská konkurence, uživatelé si mohou vybrat z velké řady softwaru, který plní požadovaný účel, proto už špičková funkčnost a spolehlivost aplikace nestačí pro udržení stávajících a získání nových uživatelů a je tedy důležité zaměřit se především na uspokojení potřeb, emoce a pocity plynoucí z používání software.

Jednotlivé aspekty, které ovlivňují uživatelský zážitek, však mohou jít proti sobě. Například bezpečnost dat jde v zásadě proti pohodlnosti při užívání. Příkladem silného zabezpečení software může být přihlašování pomocí dvou hesel a otisku prstu a odhlášení po pěti minutách neaktivity. Rozhodně se ale nejedná o řešení, které by bylo pohodlné při užívání. Je tedy důležité najít vhodný kompromis mezi protichůdnými aspekty. (Roudenský 2016)

3.4 Spolehlivost softwaru

Bezpečnost a spolehlivost jsou klíčové vlastnosti softwaru, které jsou považovány dokonce za důležitější vlastnosti než funkčnost, a to z následujících důvodů:

- Selhání softwaru může mít vliv na velké množství osob.
- Selhání softwaru vzbuzuje nedůvěru v jeho uživatelích.
- Selhání softwaru může vést ke ztrátě či odcizení informací.
- Selhání softwaru může způsobit obrovské ztráty.

Software je vždy součástí nějakého většího systému, který zahrnuje prostředí, ve kterém aplikace běží. Selhání softwaru tedy nemusí být způsobeno jen selháním sebe sama, ale také například selháním hardwaru či některé z osob, které se systémem interagují.

Spolehlivost má čtyři základní dimenze:

- *Dostupnost* (availability) znamená schopnost poskytnout požadované služby v jakémkoliv okamžiku.
- *Bezporuchovost* (reliability) znamená schopnost softwaru poskytovat služby správně a způsobem, jaký se od něj očekává.
- *Bezpečnost* (safety) vypovídá o pravděpodobnosti, s jakou software může způsobit škodu svému okolí, ať už osobám, či prostředí, ve kterém je spuštěn.
- *Zabezpečení* (security) popisuje schopnost systému bránit se náhodným nebo úmyslným vniknutím.

Dále je možné mezi dimenze spolehlivosti zahrnout také *odolnost* systému, *schopnost údržby, opravitelnost* a *zamítnutí nevalidního vstupu*. (Sommerville 2013)

Pro zajištění spolehlivosti se provádějí procesy validace a verifikace, které slouží k nalezení chyb ve specifikaci, návrhu, či samotném programu. Procesy verifikace a validace mají rovněž zjišťovat, zda software vyhovuje své specifikaci a zda odpovídá požadavkům uživatele. (Roudenský 2016)

- *Validace* znamená ověření vhodnosti nebo způsobilosti softwarového produktu pro jeho zamýšlené použití. Validační aktivity začínají již ve fázi plánování a specifikace požadavků a provázejí SW po celý životní cyklus vývoje.
- *Verifikace* slouží k ověření shody softwarového produktu s jeho specifikací. Verifikační aktivity mohou začít až po specifikaci požadavků. Začínají ve fázi návrhu a slouží ke kontrole každého produkovaného výstupu.

3.4.1 Bezporuchovost a dostupnost

Bezporuchovost a dostupnost spolu úzce souvisejí. V případě, že dojde k selhání systému, pravděpodobně nebude systém po nějakou dobu dostupný. Terminologie bezporuchovosti zahrnuje čtyři základní pojmy, kterých je vhodné se držet při rozboru spolehlivosti systému: (Sommerville 2013)

- *Lidská chyba nebo omyl* znamená selhání lidského faktoru v některé z fází vývoje či provozu softwaru. Příkladem může být rozhodnutí programátora o volbě určitého chybného postupu.
- *Systémová vada* je určitá vlastnost systému, která může způsobit systémovou chybu. Vadou systému potom může být skutečnost, že kód obsahuje chybný postup.
- *Systémová chyba* je chybný stav systému, který může způsobit jeho neočekávané chování. V případě, že je spuštěna část kódu, která obsahuje chybný postup, systém za určitých okolností může přejít do chybného stavu, i když za jiných okolností, které byly v době testování, se systém mohl chovat správně.
- *Systémové selhání* je událost, která způsobí, že systém po určitou dobu nedokáže poskytovat služby tak, jak to uživatel očekává. Systémové selhání může být způsobeno systémovou chybou.

Od bezporuchovosti softwaru se odvíjí jeho důvěryhodnost, tedy stupeň důvěry, kterou v něj mají jeho uživatelé. S tím souvisí také užitečnost – čím užitečnější je software, tím více nedostatků jsou uživatelé ochotni tolerovat. Tyto vlastnosti softwaru však není možné snadno měřit a jejich stupeň je pro každého uživatele subjektivní. Existují však metriky, které s měřením bezporuchovosti a dostupnosti mohou napomoci: (Sommerville 2013)

- *Pravděpodobnost selhání na vyžádání* (Probability of failure on demand, zkráceně POFOD), která pracuje s pravděpodobností, že požadavek na službu systému způsobí systémové selhání. Například hodnota POFOD 0,01 znamená, že k selhání při zadání požadavku dojde s pravděpodobností 1/100.
- *Frekvence výskytu selhání* (Rate of occurance of failures, zkráceně ROCOF) určuje pravděpodobné množství systémových selhání, které se objeví za určitý časový úsek.

- *Průměrná doba do selhání* (Mean time to failure, zkráceně MTTF) je opačnou hodnotou k frekvenci výskytu selhání. Pokud během jedné hodiny dojde pravděpodobně ke dvěma selháním, je ROCOF dvě selhání za hodinu a MTTF je 30 minut. MTTF by měla být několikanásobně vyšší, než je průměrný čas, během kterého uživatel se softwarem pracuje, aniž by svou práci uložil.
- *Dostupnost* (Availability, zkráceně AVAIL) udává, po jakou část zvoleného časového úseku je software funkční při zadání požadavků. AVAIL 0,99 znamená, že bude systém během jednoho dne nedostupný po dobu 14,4 minut.

3.4.2 Bezpečnost

Bezpečnost softwaru je důležitá především u kritických systémů, které mohou při svém provozu ohrozit své okolí. Většinou se jedná o systémy, které slouží k řízení určitých zařízení, ať už se jedná o stroje v továrnách či automobily nebo jiné dopravní prostředky. Ne každý typ software může svým fungováním ohrozit své okolí, například v případě kalkulačky je riziko opravdu minimální. V souvislosti s bezpečností softwaru je zavedená následující terminologie: (Sommerville 2013)

- Nehoda, případně incident, je označení pro neplánovanou událost, která způsobí újmu některé z osob, škodu na majetku či životním prostředí.
- Nebezpečí je označení pro situaci, která může způsobit nehodu nebo k ní přispět.
- Škoda je vyčíslené množství ztrát způsobených incidentem.
- Závažnost nebezpečí se popisuje různými úrovněmi (od nízké až po katastrofální) a udává hodnocení nejhorší možné škody, která může vzniknout v důsledku určitého nebezpečí.
- Pravděpodobnost nebezpečí popisuje pravděpodobnost výskytu událostí, které jsou způsobeny nebezpečím.
- Riziko je měřítko pravděpodobnosti, že systém způsobí nehodu. Riziko se odvozuje ze závažnosti a pravděpodobnosti nebezpečí a pravděpodobnosti, že nebezpečí povede k nehodě.

3.4.3 Zabezpečení

Zabezpečení systému odráží jeho schopnost odolat externím útokům, ať už náhodným či úmyslným. Se zabezpečením souvisí následující terminologie: (Sommervile 2013)

- Aktivum je něco hodnotného, co je třeba chránit před ztrátou či odcizením.
- Expozice označuje možnost ztráty či odcizení aktiva.
- Zranitelnost je označení pro slabinu počítačového systému, kterou je možné zneužít ke způsobení škody. Je možné rozlišit pět druhů zranitelnosti: (Gála 2009)
    - fyzická
    - přírodní
    - technologická
    - fyzikální
    - lidská
- Útok je označován jako bezpečnostní incident a je úmyslným zneužitím zranitelnosti systému. (Gála 2009)
- Hrozbou zabezpečení je označována možnost nežádoucího zásahu, který vede ke změně či ztrátě informací, má negativní dopad na jejich důvěrnost, integritu či dostupnost nebo působí změnu chování aplikace nebo jejích parametrů. Nejčastější hrozbou je však lidský faktor, a proto je třeba vyvíjet software tak, aby se riziko lidské chyby eliminovalo. Je možné je členit podle ovlivnitelnosti, zdroje a úmyslu (Požár 2017):
    - Podle vlivu lidského faktoru:
        - objektivní (zaviněno vnějším vlivem)
        - subjektivní (zaviněno lidským faktorem)
    - Podle zdroje:
        - vnitřní (zevnitř organizace)
        - vnější (externí)
    - Podle úmyslu:
        - náhodné
        - úmyslné
- Kontrola je ochranné opatření, které má za cíl snížit zranitelnost systému.

V případě softwaru budou aktiva pravděpodobně nehmotná v podobě dat a informací. Pokud se jedná o software, který sám o sobě slouží k zabezpečení nějakého aktiva, může být v případě nedostatečného zabezpečení ohroženo i dané aktivum. Na tento příklad se ale diplomová práce zaměřovat nebude, pouze na ochranu dat a informací. V oblasti zabezpečení a spolehlivosti systémů se často využívá zkratka CIA, pod kterou si můžeme představit pojmy Confidentiality (důvěrnost), Integrity (integrita) a Availability (dostupnost). Jedná se o klíčové vlastnosti informací, jejichž narušení vede k závažným bezpečnostním rizikům. Při plánování zabezpečení je tedy důležité se na tyto vlastnosti zaměřit v první řadě. (Čermák 2008)

- *Důvěrnost* informací je neporušena v případě, že se k informacím dostanou jen autorizovaní uživatelé. Pro každého uživatele, který má přístup do daného systému či aplikace, musí být definováno, jaké informace může číst, upravovat, či mazat.
- *Integrita* znamená správnost, úplnost a neporušenost dat a jejich zabezpečení proti neoprávněné manipulaci.
- *Dostupnost* je možné chápat jako možnost získat informace v okamžiku, kdy jsou potřeba.

3.5 Formy distribuce softwaru

Distribuce spočívá v dodání produktu uživatelům k používání. V této podkapitole budou nastíněny možnosti distribuce, jakými se software dostane k cílovým uživatelům.

3.5.1 Cena

Software může být poskytován buď zdarma, nebo se za něj platí. Za některé softwarové produkty se platí jednorázově, jiné podléhají pravidelným týdenním/měsíčním/ročním poplatkům, které se navíc mohou odvíjet od reálného využití aplikace. V případě pravidelných poplatků by měl mít uživatel k dispozici vždy aktuální verzi, u jednorázových plateb to tak být nemusí.

3.5.2 Licence

Licence definuje omezení použití softwaru, jejím cílem je chránit autorská práva vývojového týmu. Licencí existuje opravdu velké množství a není na místě tu vypisovat jejich seznam. Ke každému softwaru, i k tomu, co je poskytován zdarma, je obvykle sepisováno licenční ujednání, které většinou sepisují právníci dle specifikací vývojového týmu. Rozhodnutí o licenci však není jen na vývojovém týmu, existují zdroje (obrázky, knihovny, texty atd.), které mohou využitím v aplikaci ovlivnit podmínky distribuce softwaru.

3.5.3 Způsob

Samotná distribuce pak probíhá některým ze způsobů, které lze shrnout do dvou skupin – fyzické a internetové. Fyzicky distribuovaný software zahrnuje nejen klasický krabicový software, který se prodává v obchodech, patří sem také software přiložený ke knihám a časopisům, případně software sdílený uvnitř firmy na flash disku. Software distribuovaný prostřednictvím internetu je v současnosti nejvíce využívaný. Může se jednat o službu dostupnou prostřednictvím webového prohlížeče, případně jiné služby, či software, který si uživatelé mohou stáhnout.

3.6 Legislativní omezení

Aplikace musí být v souladu s právními předpisy, které danou oblast ovlivňují. V této práci není prostor pro popis veškerých legislativních omezení, která mohou mít na software vliv, ani to není cílem práce.

Přesto je na místě alespoň zmínit jednu z nejdiskutovanějších změn v oblasti ochrany osobních práv v online prostředí v rámci Evropské unie. Jedná se o nařízení EU 2016/679 (GDPR), které představuje právní rámec ochrany osobních údajů platný na celém území EU a začne platit 25. května 2018. Zahrnuje přísnější podmínky pro zpracování osobních údajů ve prospěch uživatelů. (Úřad pro ochranu osobních údajů 2013)

4. Vývoj aplikace pomocí zvolené technologie

V této části bude popsán vývoj aplikace na vystavování eReceptu. Na závěr bude aplikace zhodnocena s ohledem na aktuální trendy ve vývoji softwaru.

4.1 Volba technologie

Aplikace je vyvíjena pomocí frameworku od společnosti Unicorn. Unicorn Application Framework (zkráceně UAF) je referenční sada znalostí každého ICT specialisty v Unicornu. Jedná se o komplexní sadu návodů, komponent, nástrojů, dokumentace, školení, databázi znalostí a týmů podpory pro výrobu softwarových řešení. Popisuje výchozí postupy pro výrobu aplikací a ideální způsob využití všech komponent základní referenční architektury *Unicorn Mobile-First IoT-Ready Cloud Architecture*. Má za cíl vytvářet opakovaně, predikovatelně a efektivně softwarová řešení. (Kovář 2018a)

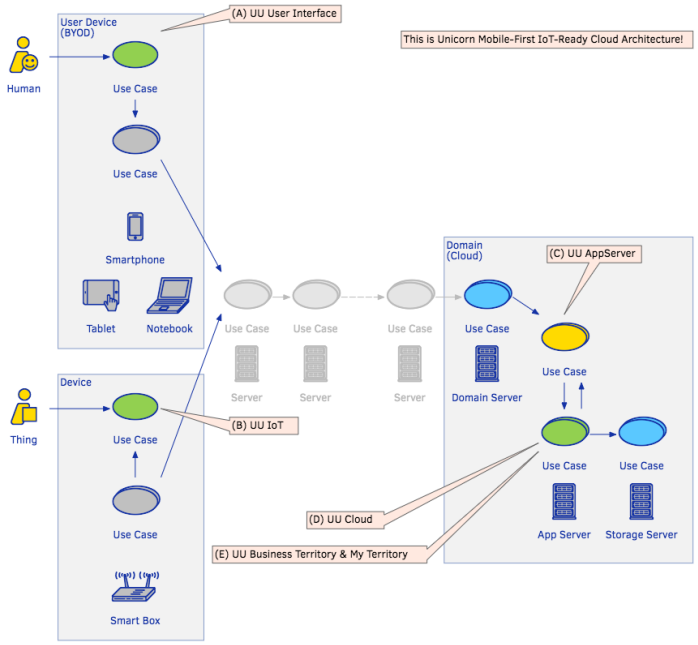
4.1.1 Use-Case Driven Approach

Přístup uplatňovaný při vývoji aplikací pomocí UAF je označován jako Use-Case Driven Approach, tedy přístup orientovaný na případy užití. To znamená, že se vždy na informační systém, resp. na aplikaci, nahlíží jako na soustavu funkčností (use-case, případy užití), která řeší nějakou problematiku. Za jednu z nejdůležitějších částí vývojového procesu je považován kontakt s uživateli a právě Use-Case Driven Approach může vzájemnou komunikaci usnadnit. Stanoví se klíčové funkčnosti, které uživatel ve své aplikaci potřebuje, a role, které je mohou spouštět. V případě, že se uživatel v oblasti ICT nepohybuje, není třeba ho zatěžovat podrobnostmi. Je však vhodné, aby uživatelé použitým technologiím alespoň trochu rozuměli, potom je snazší vyvinout produkt, který bude podle jejich představ. Slovní popis funkčností je vhodné doplnit grafickým znázorněním. Funkčnosti jsou zakreslovány pomocí jazyka Unicorn Universe Business Modeling Language (zkráceně uuBML), který vychází z UML. Ve společnosti Unicorn je zápis pomocí jazyka uuBML upřednostňován, protože je stručný a výstižný a usnadňuje pochopení problematiky. (Kovář 2018a)

4.1.2 Unicorn Mobile-First IoT-Ready Cloud Architecture

Unicorn Mobile-First IoT-Ready Cloud Architecture je základní referenční architektura společnosti Unicorn. Je navržena tak, aby pokrývala trendy minimálně na následujících pět let. Je podpořena fyzickou implementací systémových komponent ve vybraných technologiích. Jednotlivé vrstvy a komponenty referenční architektury lze použít jako jeden celek nebo jednotlivě. Skládá se z těchto částí, jejichž propojení zachycuje Obrázek 35: (Kovář 2018b)

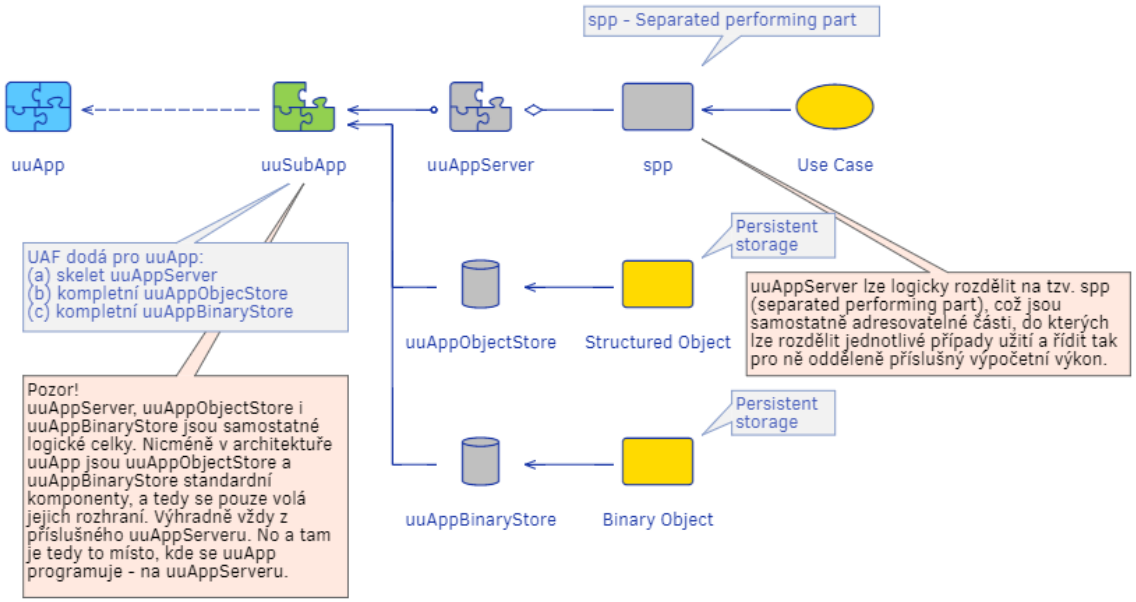
- uuUserInterface – framework, který popisuje, jak psát grafická uživatelská rozhraní pro všechna současná zařízení.
- uuIoT – framework, který popisuje, jak ovládat internet věcí.
- uuAppServer – framework, který popisuje, jak psát servery, které běží v cloudu.
- uuCloud – framework, který popisuje, jak plně automatizovaně a dynamicky ovládat nabízenou výpočetní kapacitu cloudu.
- uuBT a uuMT – popisuje, jak psát informační systémy pro organizace a jednotlivce.



Obrázek 35: Unicorn Mobile-First Iot-Ready Cloud Architecture
Zdroj: Kovář 2018a

4.1.3 uuApp

Aplikace vyvíjená pomocí UAF je označována jako uuApp a jedná se o komplexní sadu ucelených funkčností, která řeší předem specifikovanou problematiku pro uživatele. Složité aplikace se mohou skládat z logických částí, které se mohou do sebe hierarchicky vnořovat. Jednotlivé logické části jsou označovány pojmem uuSubApp – taková logická část aplikace, která se již dále z tohoto pohledu nedělí. Každá uuSubApp je relativně samostatnou jednotkou. (Kovář 2018b)



Obrázek 36: Struktura uuApp
Zdroj: Kovář 2018b

Každé uuSubApp potom přísluší právě jeden logický server označovaný jako uuAppServer. Ten je propojen s perzistentními datovými úložišti, ke kterým je možné přistupovat pouze prostřednictvím volání jednotlivých funkčností, označovaných jako uuCommands (zkráceně uuCMD), dané uuSubApp. UAF rozlišuje dvě různá datová úložiště podle druhu dat, která jsou v nich uložena:

- UuAppObjectStore slouží k ukládání strukturovaných dat. Každý uuAppServer obvykle potřebuje minimálně jeden uuAppObjectStore, protože v něm jsou uloženy informace o jednotlivých případech užití a osobách, které je mohou spouštět.
- UuAppBinaryStore slouží k ukládání binárních dat. Nemusí být přítomen v každé aplikaci.

Perzistentní objekty, které k uuSubApp přísluší, jsou seskupovány do aplikačních prostorů uuAppWorkspace, které jsou určeny jednoznačným identifikátorem (uuAppWorkspaceId, zkratka awid). Awid je součástí adresace případů užití z rozhraní aplikace. UuSubApp jsou obvykle buď tzv. jednoinstanční (uuSubApp je přidělen pouze jeden aplikační prostor), nebo víceinstanční (uuSubApp je přidělen aplikační prostor pro každou instanci jejího nasazení). (Kovář 2018b)

UuApp obvykle má také frontendovou část, která slouží k interakci s uživatelem. UAF nabízí jako svou součást uuUserInterface zahrnující frontendovou knihovnu UU5, která obsahuje sadu základních komponent a maximálně podporuje vytváření dalších opakovaně použitelných komponent. Frontendová část se serverovou částí komunikuje prostřednictvím RESTful API, může být tedy vyvíjena i pomocí jakékoli jiné technologie.

4.2 Popis řešené problematiky

K datu 1. ledna 2018 vznikla povinnost předepisovat recepty pouze elektronicky. Preskripce receptu v elektronické podobě, označovaného jako eRecept, je upravena pomocí následujících zákonů a vyhlášek (SÚKL 2017d):

- Zákon č. 378/2007 Sb., o léčivech a o změnách některých souvisejících zákonů
- Zákon č. 372/2011 Sb., o zdravotních službách a podmínkách jejich poskytování (zákon o zdravotních službách)
- Vyhláška č. 54/2008 Sb., o způsobu předepisování léčivých přípravků, údajích uváděných na lékařském předpisu a o pravidlech používání lékařských předpisů, ve znění pozdějších předpisů
- Vyhláška č. 84/2008 Sb., o správné lékárenské praxi, bližších podmínkách zacházení s léčivy v lékárnách, zdravotnických zařízeních a u dalších provozovatelů a zařízení vydávajících léčivé přípravky, ve znění pozdějších předpisů
- Vyhláška č. 236/2015 Sb., o stanovení podmínek pro předepisování, přípravu, distribuci, výdej a používání individuálně připravovaných léčivých přípravků s obsahem konopí pro léčebné použití

Datové úložiště pro sběr a zpracování elektronicky předepisovaných léčivých přípravků označované jako Centrální úložiště elektronických receptů (zkráceně CÚER) zřizuje a provozuje Státní ústav pro kontrolu léčiv (zkráceně SÚKL). Provozuje také Registr léčivých přípravků s omezením, zkráceně RLPO (SÚKL 2017e).

Vystavování eReceptu potom funguje na následujícím principu (SÚKL 2017e):

- Lékař vystaví recept, který zaregistruje do Centrálního úložiště elektronických receptů Státního ústavu pro kontrolu léčiv. Tím získá identifikátor eReceptu, což je alfanumerický kód o délce dvanácti znaků. Kód může být vyjádřen i pomocí čárového kódu či QR kódu. (SÚKL 2017a)
- Identifikátor eReceptu, případně další informace o receptu, lékař pacientovi předá pomocí některého z těchto způsobů (SÚKL 2017f):
    - Papírová průvodka
    - SMS
    - E-mail
    - Aplikace
- Pacient přijde do lékárny a lékárníkovi předá identifikátor eReceptu.
- Lékárník pomocí identifikátoru eRecept dohledá a vydá podle něj léky.
- Informaci o výdeji léku zapíše lékárník do Centrálního úložiště elektronických receptů.

Možnost vystavování eReceptu je podmíněna několika kroky, především proto, aby bylo zajištěno, že recepty vydávají pouze autorizované subjekty. Pro účast v systému elektronické preskripce je nezbytné vlastnit kvalifikovaný certifikát pro elektronický podpis. Ten v současnosti mohou vydávat tři certifikační autority – eIdentity, PostSignum a První certifikační autorita. Jeho vystavení obvykle vyžaduje osobní návštěvu certifikační autority. (SÚKL 2017c) Poté je třeba podat žádost o zřízení přístupu do Centrálního úložiště. Přístup musí být zřízen pro lékaře i pro zdravotnické zařízení. Po udělení přístupu je potřeba provést základní nastavení, jako je například změna hesla. Aby bylo možné předepisovat individuálně připravované léčivé přípravky s obsahem konopí pro léčebné účely, je třeba navíc zažádat o přístup do Registru léčivých přípravků s omezením (SÚKL 2017b).

4.3 Aplikace +4U eRecept

V této podkapitole bude stručně popsána aplikace +4U eRecept, která má za cíl umožnit lékařům vystavovat eRecepty.

Na úvod je vhodné zmínit, kdo zastupuje klíčové role. Zadavatelem aplikace je Plus4U a.s., investorem je Unicorn Universe a.s., uživateli aplikace jsou lékaři a vývojový tým představuje společnost NeatCode s.r.o.

4.3.1 Proces vývoje

Vývoj první etapy aplikace byl zahájen v průběhu června roku 2017. Na začátku nebylo známo mnoho detailů o vyvíjené aplikaci. Cílem první etapy bylo vyvinout prototyp aplikace se základními funkčnostmi. Uživateli se zobrazí formulář pro vyplnění údajů o receptu, které zahrnují informace o pacientovi a předepsaných lécích. Uživatel může vystavit obyčejný recept nebo eRecept.

A. Pokud uživatel stiskne tlačítko „Vystavit eRecept“, aplikace provede následující kroky:
   1. Zavolá API systému SÚKL a zaregistruje eRecept do CÚER, čím získá testovací kód eReceptu.
   2. Údaje eReceptu uloží do databáze.
   3. Vygeneruje náhled eReceptu, který obsahuje údaje o eReceptu a jeho jednoznačný identifikátor.
   4. Umožní eRecept stáhnout ve formátu PDF.

B. Pokud uživatel stiskne tlačítko „Vystavit recept“, aplikace provede následující kroky:
   1. Údaje receptu uloží do databáze.
   2. Vygeneruje náhled receptu, který obsahuje údaje o receptu.
   3. Umožní recept stáhnout ve formátu PDF.

V této etapě nebyl důležitý vzhled aplikace, ale bylo potřeba, aby aplikace disponovala nějakým uživatelským rozhraním, které umožní realizovat výše uvedené funkčnosti. Konkrétní návrh uživatelského rozhraní potom byl v režii vývojového týmu. Jednalo se spíše o volbu vhodné struktury aplikace, tedy použití komponent, které se hodí pro realizaci daného účelu. Styly komponent se v této etapě neřešily.

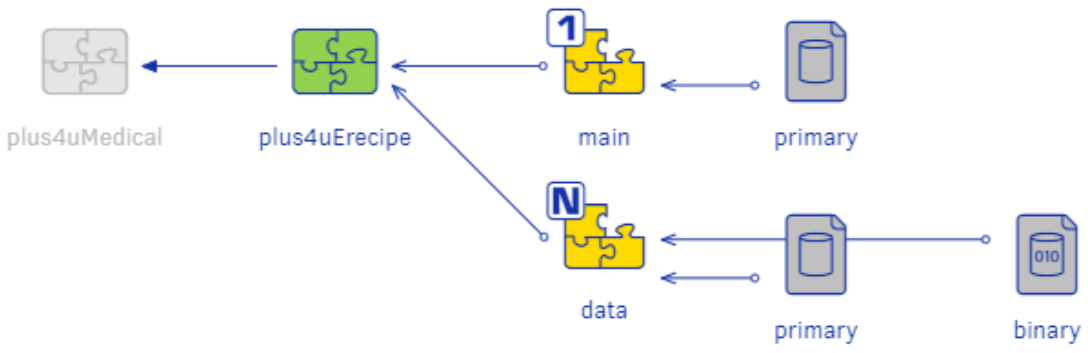
Během vývoje byl tvořen design, který popisuje aplikaci. Po vydání první etapy (která zatím nebyla určena pro uživatele) byl vytvořen návrh vzhledu jednotlivých částí aplikace. Z testování aplikace vyplynuly požadované změny a nové potřebné funkčnosti, které byly postupně přidávány do aplikace. Přidávání nových funkčností bylo realizováno na základě požadavků, které zadavatel zasílal vývojovému týmu. Postupně byly vydávány nové verze, které obsahovaly vždy sadu požadavků, a tímto postupem byl postupně tvořen produkt, který by měl respektovat potřeby uživatelů, kterými jsou především lékaři.

První verze aplikace, která byla určena pro cílové uživatele, byla oficiálně spuštěna 5. 9. 2017, tedy téměř čtyři měsíce před spuštěním povinného vystavování eReceptů. Byl tedy dostatek času na to, aby se lékaři s aplikací seznámili, vyzkoušeli si ji a na základě toho mohli podávat podněty ke zlepšení.

Aplikace od vydání prvního prototypu postupně prošla mnoha změnami, přičemž většina z nich byla do aplikace zapracována v rámci dvou sprintů, a je i nadále rozvíjena.

4.3.2 Struktura aplikace

Současná verze aplikace 1.3.3 je složena ze dvou uuSubApp – plus4uErecipe-main a plus4uErecipe-data, jak je možné vidět na obrázku níže. Každá z uvedených uuSubApp potom nabízí seznam uuCMD, které představují sadu funkčností, které umožňují realizovat případy užití. Každá z nich také rozlišuje několik rolí, které můžou spouštět určité uuCMD.



Obrázek 37: Struktura aplikace +4U eRecept
Zdroj: Měřinský 2018

Plus4uErecipe-main

Plus4uErecipe-main obsahuje uživatelský vizuální vstupní bod, pomocí kterého mohou uživatelé k aplikaci přistupovat. Jedná se o hlavní a jednoinstanční uuSubApp.

Plus4uErecipe-main rozlišuje pět základních rolí:

- *SysOwner* – systémová role, která umožňuje zřídit instanci aplikace, inicializovat ji a případně ji zrušit.
- *Authorities* – role pro nastavení práv uživatelů a jejich registraci.
- *Executives* – role pro nastavení práv nebo uživatelů.
- *Guests* – role pro všechny přihlášené uživatele.
- *Public* – role pro všechny uživatele včetně nepřihlášených.

V následující tabulce je zachycen seznam uuCmd patřící k této uuSubApp:

Tabulka 1: Plus4uErecipe-main uuCommands

Název uuCmd	Role	Popis
init	SysOwner	Inicializuje aplikaci.
registerDoctorInPlus4u	Public	Spustí registraci uživatele do služby +4U a aplikace +4U eRecept.
registerDoctor	Authorities	Dokončí registraci a uloží lékaře do databáze.
getAppInfo	Executives	Poskytne informace o Plus4uErecipe-data přidělené přihlášenému uživateli.
getMedibaseUrl	Guests	Vrátí URL adresu uuApp Plus4uMedibase, která poskytuje informace o lécích.
listDoctors	Authorities	Vrátí seznam registrovaných lékařů.
verifyReCaptcha	Public	Validuje reCaptchu při registraci nového lékaře.
updateAwidList	Authorities	Aktualizuje seznam dostupných awidů uuSubApp Plus4uErecipe-data.
deleteDoctor	Authorities	Smaže registrovaného lékaře ze systému.

Zdroj: Měřinský 2018

Grafické uživatelské rozhraní, které umožňuje lékaři k aplikaci přistupovat, spouštět funkčnosti a spravovat uložená data, bude popsáno v kapitole 4.3.3.

Plus4uErecipe-data

Plus4uErecipe-data obsahuje data jednotlivých lékařů. Každému lékaři, který se do služby registruje, je přidělen awid a založen aplikační prostor. Každá instance si ukládá data o konkrétním lékaři. UuSubApp Plus4uErecipe-main potom volá uuCMD této uuSubApp, čímž uživateli zpřístupňuje uložená data.

Plus4uErecipe-data rozlišuje čtyři základní role:

- *SysOwner* – systémová role, která umožňuje zřídit instanci aplikace, inicializovat ji a případně ji zrušit.
- *Authorities* – role pro nastavení práv uživatelů a jejich registraci.
- *Executives* – role pro nastavení práv nebo uživatelů.
- *Users* – registrovaní uživatelé, kteří mohou vytvářet recepty a eRecepty.

V následující tabulce je zachycen seznam uuCMD patřících k této uuSubApp:

Tabulka 2: Plus4uErecipe-data uuCommands

Název uuCmd	Role	Popis
init	SysOwner	Inicializuje aplikaci.
createRecipe	Executives, Users	Vytvoří recept nebo eRecept.
updateBaseInformation	Executives	Aktualizuje informace o lékaři.
updateRecipe	Executives, Users	Aktualizuje údaje o receptu nebo eReceptu.
getRecipe	Executives, Users	Vrátí informace o receptu nebo eReceptu.
listRecipes	Executives, Users	Vypíše seznam receptů a eReceptů.
sendRecipeBySms	Executives, Users	Odešle eRecept pomocí sms.
sendRecipeByEmail	Executives, Users	Odešle eRecept pomocí e-mailu.
listRecipesByInput	Executives, Users	Vypíše seznam receptů a eReceptů, které odpovídají vstupním datům.
getDoctorInfo	Executives	Vrátí informace o lékaři.
updateSmsData	Authorities	Aktualizuje konfiguraci pro odesílání sms.
getSmsData	Executives	Vrátí konfiguraci pro odesílání sms.
updateIdentificationData	Executives	Aktualizuje konfiguraci pro odesílání sms.
getAllBinaries	Executives	Vrátí seznam uložených binárních souborů.
getCertificatesValidity	Executives	Vrátí informaci o době platnosti certifikátů uložených v aplikaci.
deleteAllBinaries	Executives	Smaže uložené binární soubory.

Zdroj: Měřinský, 2018

4.3.3 Použití aplikace

Předpokládá se, že lékaři budou aplikaci využívat prostřednictvím grafického uživatelského rozhraní aplikace. To poskytuje frontendová vrstva uuSubApp Plus4uErecipe-main a je dostupné prostřednictvím webového prohlížeče. V této podkapitole se tedy práce zaměří především na použití grafického uživatelského rozhraní.

Cílovými uživateli aplikace +4U eRecept jsou lékaři, kteří díky ní mohou vystavovat eRecepty. Aplikace poskytuje dvě úrovně služeb, podle toho, zda je uživatel přihlášen a registrován. Registrace má dvě fáze, které však mohou proběhnout současně. Uživatel se musí registrovat do služby +4U, která zprostředkovává přihlášení ve všech uuApp, a do aplikace +4U eRecept, což umožní vystavování eReceptů. Také samozřejmě lékaři musí vyřídit formality spojené s vystavováním eReceptu, které jsou popsány v kapitole 4.2.

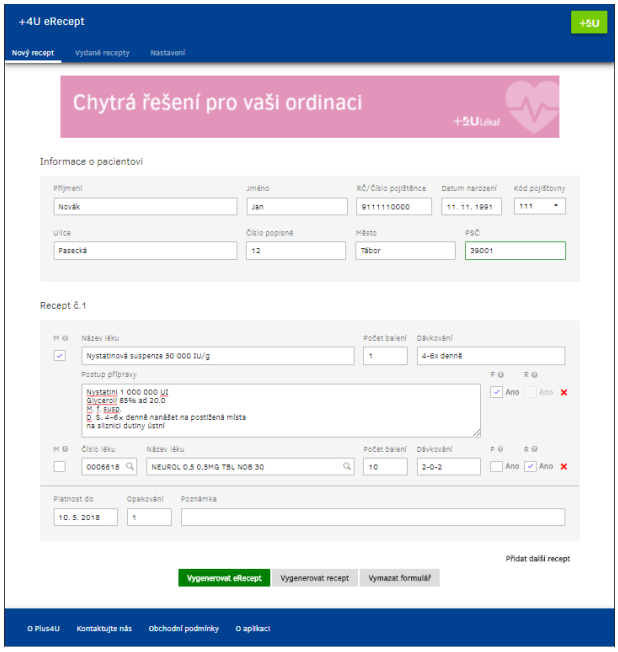
- Uživatelé, kteří nejsou v aplikaci +4U eRecept registrováni, mají k dispozici jen velmi omezené funkčnosti, které slouží k vyzkoušení aplikace. Mohou vystavit pouze obyčejný recept, který navíc obsahuje vodoznak s textem +4U eRecept. Oproti regisdtrovaným lékařům mají navíc k dispozici funkci registrovat se. Tato úroveň služeb je zdarma.
- Registrovaní a přihlášení lékaři potom mají přístup ke všem funkčnostem aplikace, které budou dále popsány. Tato úroveň je zpoplatněna částkou 49 Kč měsíčně, případně 499 Kč ročně.

Grafické uživatelské rozhraní je rozděleno do tří částí – *top* (záhlaví), *content* (obsah stránky) a *bottom* (zápatí). Tyto části se přizpůsobují skutečnosti, zda je uživatel přihlášen nebo ne.

- Záhlaví (top) umožňuje základní navigaci napříč aplikací a možnost přihlášení či odhlášení.
    - Neregistrovaný uživatel může pomocí záhlaví přepínat mezi třemi stránkami – úvod, nový recept a registrace.
    - Registrovanému a přihlášenému uživateli záhlaví nabízí přepínat mezi těmito třemi stránkami – nový recept, vydané recepty a nastavení.
- Mezi záhlavím a zápatím je zobrazen samotný obsah stránky (content). Jednotlivé stránky budou popsány v další části této podkapitoly.
- Zápatí (bottom) je stejné pro přihlášeného i nepřihlášeného uživatele a umožňuje navigaci na dvě externí stránky (o Plus4U, kontaktujte nás), a dvě stránky aplikace (obchodní podmínky a o aplikaci).

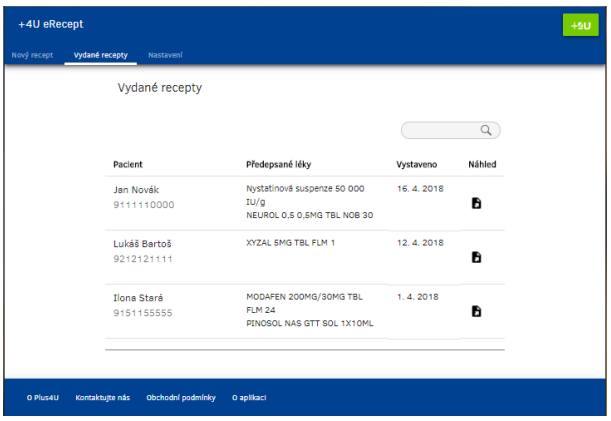
Aplikace +4U eRecept je složena z těchto stránek:

- Nový recept – obsahuje formulář pro vystavení receptu nebo eReceptu. První sekce slouží k zadání informací o pacientovi, v druhé sekci se vyplňují údaje o lécích. Počet vystavených receptů není omezen a v každém receptu je možné vyplnit jeden nebo dva léky. Na následujícím obrázku je zobrazen náhled této funkčnosti.



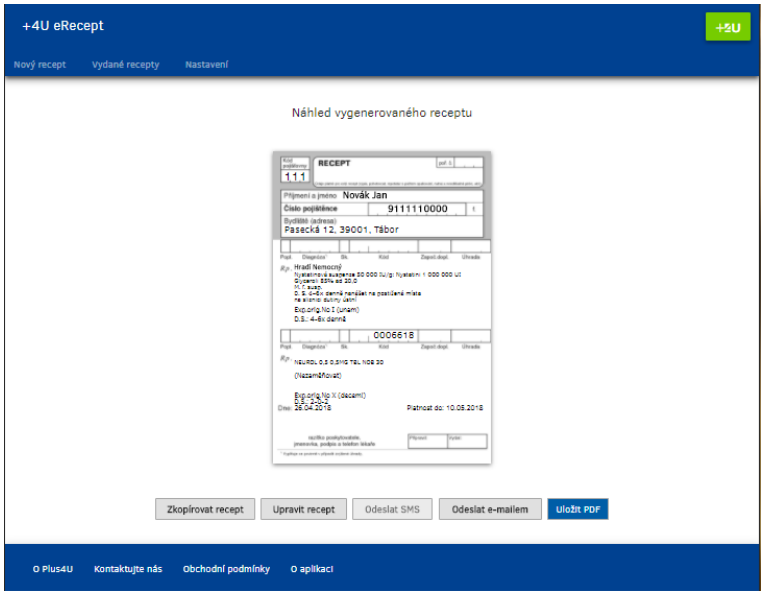
Obrázek 38: Nový recept
Zdroj: vlastní

- Vystavené recepty – zde si lékař může procházet všechny recepty, které vystavil v minulosti. Navíc je možné mezi recepty vyhledávat. Na následujícím obrázku je zachycena tato funkčnost.



Obrázek 39: Vystavené recepty
Zdroj: vlastní

- Náhled receptu/eReceptu – na tuto stránku je uživatel přesměrován po vystavení receptu či eReceptu nebo z přehledu receptů. Umožňuje spouštět další funkčnosti. Na následujícím obrázku je tato funkčnost zachycena.
    - Pokud uživatel není registrován, může z náhledu vystavit PDF, recept zkopírovat či upravit.
    - Pokud uživatel registrován je, může navíc recept odeslat e-mailem a eRecept i prostřednictvím SMS.



Obrázek 40: Náhled
Zdroj: vlastní

- Nastavení – obsahuje formulář pro vyplnění údajů o lékaři. Uživatel sem vyplní své jméno, příjmení, kontaktní údaje a dále informace potřebné pro SÚKL, jako je například kvalifikační certifikát.
- Úvod – obsahuje informace o aplikaci pro nepřihlášené uživatele.
- Obchodní podmínky – popisují podmínky použití aplikace +4U eRecept.
- O aplikaci – popisuje aplikaci, její tvůrce a použité technologie.

V budoucnu je v plánu aplikaci +4U eRecept napojit na rezervační systém +4U Lékař, který umožňuje pacientům objednat se k lékaři a lékařům efektivněji plánovat svůj čas.

4.4 Volba hodnotících metod

V předchozích kapitolách byly popsány aktuální trendy ve vývoji softwaru. Jak ale měřit, zda vyvíjená aplikace tyto trendy splňuje? V této kapitole budou shrnuty poznatky z teoretické části práce. Pomocí těchto poznatků bude následně aplikace +4U eRecept hodnocena.

Trendy ve vývoji softwaru se neustále mění. Nelze tedy najít univerzální metriku, kterou by bylo možné měřit, zda aplikace vyhovuje současným trendům, a kterou by bylo možné uplatnit dnes, za deset let i před patnácti lety. Vždy je tedy potřeba nejprve zjistit, jaké jsou aktuálně trendy ve vývoji a až poté je možné hodnotit, zda je aplikace splňuje.

V této části se autorka pokusí shrnout poznatky zjištěné během psaní diplomové práce. Skutečnosti jsou často dost zjednodušené, aby je bylo možné snadno popsat a sledovat.

1. Během vývoje je klíčová komunikace mezi vývojovým týmem, zadavatelem a uživateli.
2. Aplikace by měla být dostupná z mobilních zařízení.
3. Aplikace by měla zohlednit skutečnost, že uživatelé obvykle mají více různých zařízení, a měla by zpřístupnit ta samá data z jakéhokoliv zařízení.
4. Uživatelé neradi instalují aplikace do svých zařízení, proto často dávají přednost aplikacím, které instalaci nevyžadují a které nemusí pravideně aktualizovat.
5. Využití internetu v aplikaci může přinášet mnoho výhod, ale nevýhodou je, pokud je na něm aplikace plně závislá.
6. Aplikace musí být dohledatelná prostřednictvím webových vyhledávačů při zadání vhodných klíčových slov, k čemuž se často využívá SEO. A to platí nejen pro webové aplikace. Nativní aplikace sice nejsou dostupné prostřednictvím webových prohlížečů, ale přesto je dobré, aby o nich na internetu byla alespoň zmínka. SEO je potom možné uplatnit například pro stránku, která o existenci aplikace informuje, či nabízí odkaz k jejímu stažení.
7. V současnosti stále stoupá množství různých zařízení, která jsou propojena sítí Internet of Things, a při vývoji aplikace je vhodné se na taková zařízení zaměřit, pokud to má pro její funkčnosti smysl.

8. Je efektivní co největší část aplikace provozovat v cloudu, který umožňuje přidělovat výpočetní výkon podle aktuálního vytížení. Pokud je třeba určitou část dat chránit, je možné využít hybridní cloud.
9. Virtuální, rozšířená či hybridní realita přináší mnoho nových možností interakce s uživatelem. Její vývoj je však značně nákladný a není dostupná každému.
10. Využití biometrik pro zabezpečení aplikace je účinné a pro uživatele pohodlné.
11. Umělá inteligence je v současnosti stále více využívána v mnoha oblastech vývoje softwaru. Dokáže rozpoznat emoce uživatelů, používat přirozený jazyk, vyvozovat z dat nové skutečnost a díky tomu například předpovídat, co chce uživatel udělat. V současnosti je rozšířená především ve formě chatbotů.
12. Firmy objevují výhody principu blockchainu a začínají je využívat i pro další oblasti.
13. V současnosti již každý vývojový tým využívá vývojová prostředí a další nástroje, které usnadňují vývoj, jako například verzovací systémy.
14. Pokud má být aplikace dostupná třetím stranám, je vhodné poskytovat API nezávislé na platformě a programovacím jazyce.
15. Dochází k souboji mezi objektově orientovaným a procedurálním programováním. S přístupy založenými na přijetí změn se nemusí detailní návrh objektů jevit jako dobrý nápad.
16. Především ve fázích návrhu a implementace je vhodné využívat modelovací jazyky. Jejich využití k popisu skutečnosti je stručnější, výstižnější než samotný text, snadno pochopitelné a navíc mohou umožnit rovnou vygenerovat základ zdrojového kódu. Přesto však je vhodné modely doplnit i slovním popisem.
17. Nové druhy aplikací umožňují propojit webové a nativní aplikace tak, aby získaly výhody webové aplikace, ale neztratily výhody nativní aplikace (dostupnost offline, snadný přístup k hardwaru zařízení, možnost stáhnout aplikaci v obchodu určeném pro dané zařízení, atd.).
18. Mezi současně nejpoužívanější programovací jazyky patří Java, JavaScript, C a jazyky z něj odvozené, PHP a Python. Tyto jazyky jsou na trhu již déle, jsou ověřeny praxí, je však otázkou, jak dlouho se ještě na hlavních příčkách udrží. Je vhodné zvolit jazyk, který je často používaný, protože je možné čerpat ze zkušeností jiných, je snazší sehnat vývojáře, kteří takový jazyk umí, a je větší šance, že tento programovací jazyk bude i nadále podporován.

19. Relační databáze v současnosti stále vedou nad NoSQL databázemi, nejoblíbenější je MySQL. Nehodí se však pro všechny druhy aplikací. NoSQL databázím momentálně vládne MongoDB.
20. Data jsou v současnosti jedním z nejdůležitějších aktiv firem. Neslouží jen k uchovávání často velmi důležitých informací, ale mohou být dále analyzována pro vyhodnocování nových závěrů a předvídání dalšího vývoje.
21. Více než dvě třetiny vývojových týmů využívají agilní přístupy. Některé části vývojového procesu mohou být prováděny tradičně, pokud je například potřeba detailní návrh před zahájením implementace. Přesto je možné využít agilní přístup pro ostatní fáze vývoje. Tradiční metodiky lze kombinovat s agilními a vybrat si z každého přístupu to, co je pro vývoj dané aplikace nejvhodnější.
22. Vývoj softwaru je možné řídit jako projekt, ale s tím rozdílem, že kvalita je neměnným faktorem a pokud není možné stihnout práci v dohodnutém termínu a s dohodnutým rozpočtem, vývojový tým by se měl dohodnout se zadavatelem na snížení rozsahu poskytovaných funkčností. Je lepší na méně důležitou funkci počkat, než mít aplikaci kompletní, ale nespolehlivou či dokonce nefunkční.
23. Velmi důležité je sdílení znalostí uvnitř vývojového týmu a mezi vývojovým týmem a dalšími klíčovými rolemi. Jen tak je možné efektivně vyvíjet software, který bude cílovým uživatelům vyhovovat.
24. Cloudové služby představují efektivní možnost, jak zajistit spolehlivost aplikace, aniž by musel vývojový tým budovat serverovou infrastrukturu. Při plánování softwarové architektury je třeba určit, jaké části budou tvořit cloudové služby.
25. Software musí vycházet z požadavků uživatelů. Pokud nejsou zohledněny, nemá takový vývoj smysl.
26. Uživatelé dávají při výběru přednost kvalitě, a to platí i u softwarových produktů. Kvalita neznamená jen funkčnost, zahrnuje vlastnosti jako je uživatelská přívětivost aplikace, spolehlivost, její zabezpečení a mnoho dalšího.
27. User experience je v dnešní době nutnost. Na trhu softwaru existuje obrovská konkurence a s tou se dá bojovat jedině tak, že bude uživatel s aplikací spokojen natolik, že nebude mít zapotřebí hledat alternativy. Každý uživatel má jiné priority, pro někoho není důležité zabezpečení, ale zato nesnáší nepřehledné aplikace, jiní uživatelé odsuzují jednoduché aplikace, které jim nedávají možnost volby.

28. V současnosti nejoblíbenější forma distribuce je taková, kdy se uživatel nemusí o nic starat. Nelze předpokládat, že si bude uživatel v dnešní době chodit kupovat každý měsíc novou verzi krabicového softwaru. Vzhledem k tomu, kolik služeb je dostupných prostřednictvím internetového prohlížeče, má aplikace, která vyžaduje instalaci samotného programu a poté obvykle další instalace aktualizací, malou šanci na úspěch. Přesto se najdou uživatelé, kteří především u aplikací, které pracují s citlivými daty, upřednostňují uchovávat data na svém disku.
29. Cena aplikace musí být pro uživatele přijatelná, jako tomu je u jiných produktů.
30. Licenční podmínky by měly být v souladu s právními předpisy, chránit autorská práva vývojového týmu a na druhou stranu neomezovat uživatele.

Nyní je třeba z těchto poznatků nějakým způsobem vypracovat systém, pomocí kterého lze aplikace hodnotit. Autorka této práce navrhuje, že by takové hodnocení mohlo být realizováno pomocí sady popisných charakteristik aplikace, u kterých je možné jasně uvést, zda je aplikace splňuje, či nesplňuje. Nyní tedy budou z nalezených poznatků vybrány požadavky, které je možné transformovat na popisnou charakteristiku aplikace.

Pro aplikaci, která je v souladu se současnými trendy ve vývoji software, podle autorky práce platí tyto charakteristiky:

- Během vývoje aplikace probíhá komunikace mezi vývojovým týmem a ostatními zúčastněnými stranami.
- Aplikace je dostupná z mobilních zařízení.
- Uživatelské rozhraní aplikace je přizpůsobeno mobilnímu zobrazení.
- Aplikační data jsou dostupná z různých zařízení nebo je alespoň zajištěna možnost jejich přenosu.
- Aplikaci není nutné instalovat.
- Aplikace využívá výhod internetu.
- Aplikace, nebo její částečná verze, je dostupná offline.
- Aplikace uplatňuje Search Engine Optimalization.
- Aplikace dokáže komunikovat se zařízeními v síti IoT.
- Aplikace nebo její část je provozována v cloudu.
- Aplikace využívá výhod VR, AR nebo MR.

- Aplikace umožňuje přihlášení či idenfikaci uživatele pomocí biometrik.
- Aplikace využívá výhod, které přináší umělá inteligence.
- Vývoj aplikace je podpořen vhodnými softwarovými nástroji.
- Aplikace poskytuje API nezávislé na platformě či programovacím jazyce.
- Aplikace je vyvíjena pomocí vhodně zvoleného objektově orientovaného, případně procedurálního programovacího jazyka.
- Při vývoji aplikace jsou používány modelovací jazyky.
- Aplikační data jsou dále analyzována pro zjišťování nových poznatků, které mohou přinést konkurenční výhodu.
- Při vývoji aplikace se alespoň částečně používají agilní metodiky.
- Klíčové znalosti jsou sdíleny vhodnou formou napříč vývojovým týmem.
- Aplikace je vyvíjena na základě požadavků zadavatele či uživatelů.
- Aplikace je uživatelsky přívětivá, kvalitní a spolehlivá.
- Údržbu aplikace je možné provádět bez nutnosti zásahu uživatelů.

Přestože byly poznatky dost zobecněny, stejně nemohou platit pro všechny aplikace za všech situací. Při hodnocení konkrétní aplikace je tedy možné nějaké body vynechat, pokud pro ni nemají smysl, případně doplnit jiné.

Základem hodnocení je tedy určit, které body aplikace splňuje a které nikoli. U každého bodu je vhodné hodnotit jej i slovně a zaměřit se především na možnosti zlepšení. Nepředpokládá se, že aplikace jednoznačně splní všechny body, téměř vždy je prostor ke zlepšení. Je však třeba upozornit, že takové hodnocení bude vždy v určité míře subjektivní, jen některé z charakteristik lze nějakým způsobem přesně vyčíslit.

I tak je ale možné definovat systém číselného hodnocení. Zda aplikace splňuje jednotlivé body lze vyjádřit pomocí 0 (neplňuje) a 1 (splňuje), případně pomocí stupnice o zvoleném počtu stupňů, například 0 (nesplňuje), 1 (spíše nesplňuje), 2 (nelze určit), 3 (spíše splňuje) a 4 (splňuje). Potom lze vyjádřit hodnocení jako poměr počtu získaných bodů vůči celkovému počtu dosažitelných. Podobně lze taky jednotlivým charakteristikám definovat číselně váhu, pokud je potřeba některé body zdůraznit nad jinými. Získané body pro jednotlivé charakteristiky se pak navíc vynásobí jejich váhou.

Cílem tohoto hodnocení však podle autorky této práce není vyčíslit míru, v jaké aplikace vyhovuje současným trendům, ale zjistit, co je možné vylepšit, aby se tato míra zvýšila.

4.5 Zhodnocení aplikace a procesu vývoje

Nyní bude systém hodnocení navržený v předchozí podkapitole použit pro hodnocení aplikace +4U eRecept. Každý jednotlivý bod bude ohodnocen podle toho, zda jej aplikace +4U eRecept splňuje. Rozlišovány budou tři stupně: nesplňuje (N), možnost zlepšení (M) a splňuje (S).

Tabulka 3: Hodnocení aplikace +4U eRecept

1. Během vývoje aplikace probíhá komunikace mezi vývojovým týmem a ostatními zúčastněnými stranami.	M
2. Aplikace je dostupná z mobilních zařízení.	S
3. Uživatelské rozhraní aplikace je přizpůsobeno mobilnímu zobrazení.	M
4. Aplikační data jsou dostupná z různých zařízení nebo je alespoň zajištěna možnost jejich přenosu.	S
5. Aplikaci není nutné instalovat.	S
6. Aplikace využívá výhod internetu.	S
7. Aplikace, nebo její částečná verze, je dostupná offline.	N
8. Aplikace uplatňuje Search Engine Optimalization.	M
9. Aplikace dokáže komunikovat se zařízeními v síti IoT.	S
10. Aplikace nebo její část je provozována v cloudu.	S
11. Aplikace využívá výhod VR, AR nebo MR.	N
12. Aplikace umožňuje přihlášení či idenfikaci uživatele pomocí biometrik.	N
13. Aplikace využívá výhod, které přináší umělá inteligence.	M
14. Vývoj aplikace je podpořen vhodnými softwarovými nástroji.	M
15. Aplikace poskytuje API nezávislé na platformě či programovacím jazyce.	S
16. Aplikace je vyvíjena pomocí vhodně zvoleného objektově orientovaného, případně procedurálního programovacího jazyka.	S
17. Při vývoji aplikace jsou používány modelovací jazyky.	S
18. Aplikační data jsou dále analyzována pro zjišťování nových poznatků, které mohou přinést konkurenční výhodu.	S
19. Při vývoji aplikace se alespoň částečně používají agilní metodiky.	S
20. Klíčové znalosti jsou sdíleny vhodnou formou napříč vývojovým týmem.	S
21. Aplikace je vyvíjena na základě požadavků zadavatele či uživatelů.	S
22. Aplikace je uživatelsky přívětivá, kvalitní a spolehlivá.	S
23. Údržbu aplikace je možné provádět bez nutnosti zásahu uživatelů.	S

Zdroj: vlastní

Nyní je třeba jednotlivé body okomentovat a najít možnosti zlepšení.

1. Během vývoje probíhají pravdelné schůzky se zadavatelem, na kterých je informován o současném postupu, případných problémech a na kterých upřesňuje své požadavky. S uživateli komunikuje především produktový specialista Milan Brašna. Tento bod byl hodnocen jako „možnost zlepšení“, protože se autorka domnívá, že by aplikaci prospělo rozšíření o část dostupnou pro pacienty. Ti by pak stejně jako lékaři měli možnost zobrazit přehled receptů, náhled receptu a recept vytisknout. Případně by bylo možné navíc doplnit funkčnost, pomocí které by uživatel mohl zažádat o opakované vystavení receptu. Lékař by pak měl možnost zobrazit seznam žádostí o recepty a u každé položky potom možnost vystavit recept (což by přesměrovalo na předvyplněný formulář nový recept) a zrušit žádost. Také by bylo možné přidat evidenci pacientů. To by lékaři usnadnilo vyplňování údajů o pacientovi na recept. Stačilo by zadat jeho jméno a zbytek by se doplnil automaticky.
2. Aplikace je dostupná z mobilních zařízení. Unicorn Mobile-First IoT-Ready Cloud Architecture tento trend zohledňuje a při návrhu aplikace jsou mobilní zařízení na prvním místě.
3. Vzhled stránky se přizpůsobuje desktopovým zařízením, tabletům a novějším mobilním telefonům. Autorka práce narazila na drobnou nedokonalost – na nejstarších mobilních telefonech s nízkým rozlišením může být na výšku část stránky skryta, avšak neomezuje to funkčnosti a při zobrazení na šířku se stránka zobrazí správně.
4. Každý uživatel má po přihlášení prostřednictvím služby Plus4U přístup k datům aplikace z různých zařízení.
5. Aplikaci není nutné instalovat, je však třeba zaregistrovat se a vyřídit administrativu spojenou s vystavováním eReceptu.
6. Aplikace je dostupná prostřednictvím webového prohlížeče na adrese *https://www.plus4u.net/erecept*. Je provozována v cloudu, data jsou prostřednictvím internetu sdílena mezi všemi zařízeními, které uživatel používá.
7. Aplikace není dostupná offline. Mobilní prohlížeč sice dokáže zobrazit určitou offline verzi, ale vzhled se nepřizpůsobí mobilnímu zařízení. K fungování však vyžaduje přístup k dazabázi. Je samozřejmé, že vystavovat eRecepty bez internetového připojení nelze, ale bylo by možné například procházet již vystavené. Závislost na

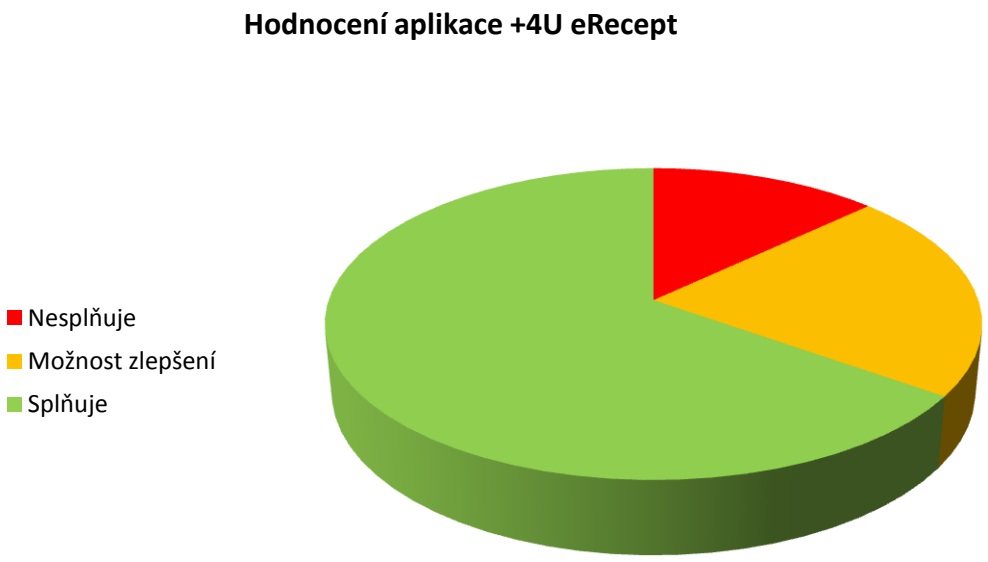
internetovém připojení může také vést ke ztrátě dat v případě, že dojde k náhlému výpadku internetového připojení.

8. Search Engine Optimalization není zvládnuta úplně nejlépe. Aplikace +4U eRecept se při vyhledávání klíčového slova „eRecept“ objeví až na dalších stránkách. Existují ale i jiné cesty, jakými se lékaři o aplikaci mohou dozvědět, a to je spíše otázka marketingu než vývoje softwaru.
9. Unicorn Mobile-First IoT-Ready Cloud Architecture podporuje komunikaci se zařízeními připojenými do IoT. Autorka práce navrhuje možnost zpřístupnění aplikace na chytrých hodinkách. Vzhledem k malému displeji by bylo velice nepraktické na hodinkách vystavovat recepty. Po rozšíření aplikace o část pro pacienty by však hodinky byly praktickým prostředkem pro předání kódu eReceptu. Lékaři by se například mohly zobrazovat notifikace o nových žádostech. Je však otázka, zda by bylo rozšíření na chytré hodinky pro uživatele natolik užitečné, aby se vyplatil jeho vývoj.
10. Aplikace běží v cloudu, který umožňuje přidělovat výpočetní prostředky podle výkonu aplikace.
11. Aplikace nedisponuje VR, MR ani AR, podle autorky práce však využití těchto technologií nemá pro aplikaci na vystavování receptů příliš velký smysl.
12. Aplikace vyžaduje pro plnou funkčnost přihlášení uživatele do služby Plus4U. Ta sice umožňuje přihlášení přes různé sociální sítě, přihlášení pomocí biometriky však momentálně možné není.
13. Funkčnosti aplikace nejsou založeny na zpracování pomocí umělé inteligence. Vzhledem k množství textu, které musí lékař ve formuláři vyplnit, by mohlo být efektivní umožnit zadávání informací hlasem, k čemuž se obvykle využívá umělá inteligence. To je však dostupné v mobilních zařízeních většinou automaticky. Dále by bylo možné do aplikace přidat chatboty pro komunikaci s lékaři, řešení problémů a hledání možností zlepšení.
14. Při vývoji je doporučené vývojové prostředí RubyMine od JetBrains. Zdrojový kód jednotlivých verzí aplikace je spravován pomocí verzovacího nástroje Git. Bylo by však vhodné vývoj zautomatizovat pomocí nástrojů, které by umožnily design nebo jeho část převést do zdrojového kódu.

15. Jednotlivé uuCommands aplikace jsou přístupné prostřednictvím RESTful API. Může tedy komunikovat s jinými uuApp i dalšími druhy aplikací.
16. Frontend aplikace je vyvíjen pomocí JavaScriptu, který využívá knihovny React, uu5, uuPlus4U5 a další. Serverová část je napsaná pomocí Ruby, do budoucna však uuAppServer bude vyvíjen pomocí Node.js.
17. Při návrhu aplikace se využívá modelovací jazyk uuBML, který je odvozen z UML.
18. Aplikační data je možné dále analyzovat, součástí aplikace je Google Tag Manager, který umožňuje přidávání dalších nástrojů, které mohou sloužit například pro měření návštěvnosti. Provoz serverové části je zaznamenáván do logů. Mnoho dat také jistě plyne z využití cloudu. Data, která je možné dále vyhodnocovat, tedy k dispozici jsou, jejich další zpracování však již není otázkou vývojového týmu.
19. Přístup k vývoji je možné považovat za agilní. Cílem vývojového týmu je dodávat software, který vyhovuje potřebám uživatelů. Proces vývoje je založen na komunikaci mezi vývojovým týmem, zadavatelem, případně uživateli. Pravidelně jsou dodávány nové verze.
20. Všechny důležité znalosti jsou zaznamenávány prostřednictvím služby Plus4U a knih v aplikaci uuDocKit. Dále probíhají školení a open sessions, na kterých jsou představovány novinky. Vývojový tým se pravidelně schází a předává si informace o současném stavu projektů, na kterých tým pracuje.
21. Nové funkčnosti jsou do aplikace přidávány na základě specifikovaných požadavků, obvykle jako sada několika nových funkčností v rámci jednotlivých sprintů.
22. Tento bod je nejhůře měřitelný. Uživatelské rozhraní je navrhováno tak, aby byl maximalizován uživatelský požitek. Aplikace splňuje svou specifikaci a poskytuje možnost vystavit recept i eRecept a následně ho vytisknout a poslat e-mailem či prostřednictvím sms. Testování probíhá nejprve pomocí jednotkových testů, potom probíhá FAT a následně UAT v akceptačním prostředí. Pokud projde všemi testy, je nasazen do provozního prostředí. Aplikace prošla procesem certifikace, během kterého jsou všechny aplikace vyvíjené v rámci Unicornu kontrolovány a pomocí modelu FURPS je ověřována jejich kvalita. Kvalita kódu je také kontrolována pravidelnými revizemi uvnitř vývojového týmu. 24 hodin denně je dostupný +4U Helpdesk, který řeší případné problémy. Podle autorky je tedy možné aplikaci považovat za spolehlivou a kvalitní.

23. Vzhledem k tomu, že je aplikace dostupná prostřednictvím webového prohlížeče, pro změny v aplikaci není nutný zásah uživatelů. Díky rychlému nasazení nové verze, většinou v rozsahu několika minut, nedochází k dlouhodobé nedostupnosti aplikace.

Za předpokladu, že by všechny body měly stejnou váhu, je možné výsledek hodnocení zobrazit pomocí následujícího grafu. Je však nepravděpodobné, že například využití virtuální reality v aplikaci pro vystavování receptů bude stejně důležité jako její dostupnost z mobilních zařízení. Zvolení vhodných vah pro každý bod by však vyžadovalo mnohem podrobnější analýzu, na kterou v této práci již není prostor.



Obrázek 41: Hodnocení aplikace +4U eRecept
Zdroj: vlastní

Na základě výsledků je možné říci, že Unicorn Application Framework umožňuje budovat aplikace v souladu se současnými trendy a aplikace +4U eRecept efektivně využívá jeho možnosti.

Závěr

Obsahem této práce bylo hledání současných trendů v oblasti vývoje softwaru. Jedná se o velmi širokou oblast, proto se autorka této práce zaměřila na aspekty, které považuje za klíčové při vývoji jednoúčelových aplikací pro jednotlivce či menší firmy.

Mezi hlavní současné trendy ve vývoji aplikací patří využití internetu a hlavně cloudových služeb, které pomáhají zajistit spolehlivost a dostupnost aplikace, efektivitu, a to hlavně tím, že dokážou škálovat výpočetní výkon a především vývojovému týmu umožňují zaměřovat se na důležité – vývoj funkčností. Aplikace musí být dostupné z mobilních zařízení a respektovat skutečnost, že uživatelé k aplikacím přistupují obvykle z více různých zařízení a aplikace by jim měla vždy zpřístupnit jejich data. Při vývoji se prosazují především agilní přístupy, protože se potřeby uživatelů, konkurenční prostředí i dostupné možnosti rychle vyvíjejí a těmto změnám je potřeba se přizpůsobit. Existují nové druhy aplikací, například progresivní webové aplikace, které využívají výhod webových a nativních aplikací a zároveň eliminují jejich nevýhody. Kvalita softwaru je velmi důležitá, zahrnuje mnoho dalších aspektů a je těžké ji měřit, ale pro uživatele ji představuje především User Experience. Vývoj aplikace se neobejde bez komunikace mezi vývojovým týmem, zadavatelem a uživateli.

Poznatky o současných trendech byly zobecněny do několika bodů, pomocí kterých je podle autorky této práce možné hodnotit, zda aplikace vyhovuje současným trendům. Jak již bylo zmíněno, v této práci nebylo možné zkoumat všechny aspekty, které s vývojem softwaru souvisí, například marketing, či datovou analýzu. Proto je možné doplňovat další charakteristiky, případně odebrat ty, které nejsou pro danou aplikaci důležité. Postup hodnocení však vždy bude subjektivní a jeho hlavním cílem není vyčíslit určitou hodnotu, která by popisovala míru souladu se současnými trendy, ale najít možnosti zlepšení.

Zobecněné poznatky byly následně aplikovány pro hodnocení aplikace +4U eRecept. Z hodnocení vyplynulo, že aplikace využívá dostupných možností, ale byly nalezeny možnosti zlepšení. Některé by byly značně nákladné a v současnosti i těžko uskutečnitelné, přesto ale byly nalezeny tři realizovatelné možnosti zlepšení:

- Webovou aplikaci by bylo možné proměnit na progresivní webovou aplikaci, čímž by se zpřístupnila i offline, což by mimo jiné mohlo eliminovat riziko ztráty dat v případě výpadku internetového připojení.
- Aplikaci by bylo vhodné rozšířit o část přístupnou pacientům. Uložená data by nyní sloužila oběma stranám, lékař by měl svůj seznam pacientů, ze kterých by vybíral, a nemusel by vypisovat jejich údaje ke každému receptu znovu.
- Autorka této práce by dále doporučila upravit nedostatek v zobrazení na starších telefonech. Uživatelů přistupujících z takových zařízení sice nemusí být moc, ale neesteticky působící aplikace by je mohla odradit.

Cílem této práce bylo najít současné trendy a ty následně použít k hodnocení aplikace vyvíjené pomocí zvolené technologie. Aplikování poznatků popsaných v teoretické části vedlo k nalezení realizovatelných možností zlepšení. Autorka věří, že je postup možné aplikovat i na jiné aplikace, cíl práce je tedy možno považovat za splněný.

Je však třeba zdůraznit, že hodnocení vždy bude do jisté míry subjektivní a že je potřeba jednotlivé body aktualizovat podle současného rozvoje dostupných možností, případně doplnit další body, které jsou pro danou aplikaci důležité. Jednotlivé body by rovněž bylo vhodné doplnit o váhy, které je ale třeba volit pro každou aplikaci jinak, a v této práci již nezbyl prostor pro hodnocení důležitosti jednotlivých kritérií u aplikace +4U eRecept.

Seznam použité literatury

3 základní druhy mobilních aplikací: webová, nativní či hybridní mobilní aplikace? [online], 2017. IQUEST. [cit. 2018-04-30]. Dostupné z: http://blog.iquest.cz/2017/08/3-zakladni-druhy-mobilnich-aplikaci.html

AL-BADAREEN, Anas Bassam, Mohd Hasan SELAMAT, Marzanah A. JABAR, Jamilah DIN a Sherzod TURAEV, 2011. Software Quality Models: A Comparative Study. *Software Engineering and Computer Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, 46-55. Communications in Computer and Information Science. DOI: 10.1007/978-3-642-22170-5_4. ISBN 978-3-642-22169-9. Dostupné také z: http://link.springer.com/10.1007/978-3-642-22170-5_4

AVISON, D. E. a Gholamreza. TORKZADEH, c2009. *Information systems project management*. Los Angeles: SAGE. ISBN 978-1-4129-5702-1.

BRIDGWATER, Adrian, 2015. *What's The Difference Between A Software Product And A Platform?* [online]. Forbes Media [cit. 2018-04-30]. Dostupné z: https://www.forbes.com/sites/adrianbridgwater/2015/03/17/whats-the-difference-between-a-software-product-and-a-platform/#24deede656a6

BURRIS, Eddie, 2007. *Software Quality Management* [online]. [cit. 2018-04-30]. Dostupné z: http://sce2.umkc.edu/BIT/burrise/pl/software_quality_management/

CHRISTENSSON, Per., 2006. Computer Definition. *TechTerms* [online]. Sharpened Productions [cit. 2018-04-29]. Dostupné z: https://techterms.com/definition/computer

Co je cloud computing?: Průvodce pro začátečníky [online], 2018a. MICROSOFT. [cit. 2018-04-30]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-cloud-computing/

Co je to identifikátor eReceptu, 2017a. *Elektronická preskripce* [online]. Praha: SÚKL [cit. 2018-04-30]. Dostupné z: https://www.epreskripce.cz/co-je-identifikator-ereceptu

Colorblind Web Page Filter [online], 2017. TOPTAL, LLC. [cit. 2018-04-30]. Dostupné z: https://www.toptal.com/designers/colorfilter

COTS (Commercial off-the-shelf), 2016. *ManagementMania* [online]. [cit. 2018-04-29]. Dostupné z: https://managementmania.com/cs/cots-commercial-off-the-shelf

ČÁPKA, David, 2018. *UML* [online]. itnetwork.cz [cit. 2018-04-30]. Dostupné z: https://www.itnetwork.cz/navrh/uml

ČESKÝ STATISTICKÝ ÚŘAD [ČSÚ]. B Domácnosti. In: Český statistický úřad. *Informační technologie v domácnostech a mezi jednotlivci* [online]. ©2016. [cit. 2018-04-29]. Dostupné z: https://www.czso.cz/csu/czso/domacnosti_a_jednotlivci

ČERMÁK, Miroslav. CIA: Důvěrnost-Integrita-Dostupnost. *CleverAndSmart – ICT management* [online]. 2008 [cit. 2018-04-29]. Dostupné z: http://www.cleverandsmart.cz/duvernost-integrita-dostupnost/

ČERNÝ, Jan, 2017. Data, informace a cesta ke znalostem. *Informační gramotnost* [online]. [cit. 2018-04-29]. Dostupné z: https://www.informacnigramotnost.cz/data-informace-znalosti/

Developer Survey Results 2017 [online], 2017. STACKOVERFLOW. [cit. 2018-04-30]. Dostupné z: https://insights.stackoverflow.com/survey/2017

Difference Between Client-Server and Peer-to-Peer network [online], 2017. TECHDIFFERENCES. [cit. 2018-04-30]. Dostupné z: https://techdifferences.com/difference-between-client-server-and-peer-to-peer-network.html

DOVEY, Dana, 2018. *The Future of Technology Is Uncertain as Moore's Law Comes to an End* [online]. [cit. 2018-04-29]. Dostupné z: http://www.newsweek.com/future-technology-uncertain-moores-law-comes-end-807546

DUCHO, Pavel, 2014. *Verzovací systém a webintegrační projekty* [online]. LundeGaard [cit. 2018-04-30]. Dostupné z: http://www.web-integration.info/cs/blog/verzovaci-system-a-webintegracni-projekty/

EVANS, Bob, 2017. *The Top 5 Cloud-Computing Vendors: #1 Microsoft, #2 Amazon, #3 IBM, #4 Salesforce, #5 SAP* [online]. Forbes Media [cit. 2018-04-30]. Dostupné z: https://www.forbes.com/sites/bobevans1/2017/11/07/the-top-5-cloud-computing-vendors-1-microsoft-2-amazon-3-ibm-4-salesforce-5-sap/#4ac283c56f2e

Expense Manager [online], VAADIN. [cit. 2018-04-30]. Dostupné z: https://demo.vaadin.com/expense-manager/

GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ, 2009. *Podniková informatika*. 2., přeprac. a aktualiz. vyd. Praha: Grada. Expert (Grada). ISBN 978-80-247-2615-1.

GDPR (obecné nařízení) [online], 2013. Praha: Úřad pro ochranu osobních údajů [cit. 2018-04-30]. Dostupné z: https://www.uoou.cz/gdpr-obecne-narizeni/ds-3938/p1=3938

GOLDSTEIN, Alexis, Louis LAZARIS a Estelle WEYL, 2015. *HTML5 & CSS3 for the real world*. Second edition. Collingwood: SitePoint. ISBN 978-0-9874674-9-2.

HAMMINK, John, 2018. *The Types of Modern Databases* [online]. [cit. 2018-04-30]. Dostupné z: https://dzone.com/articles/the-types-of-modern-databases

HOLUB, Tomáš, 2012. *Čím se liší C# od C++* [online]. CN Invest [cit. 2018-04-30]. Dostupné z: https://www.zive.cz/clanky/cim-se-lisi-c-od-c/sc-3-a-107869/default.aspx

IDAN, Henn, 2016. *The Top 100 Java Libraries in 2016 - After Analyzing 47,251 Dependencies* [online]. [cit. 2018-04-30]. Dostupné z: https://blog.takipi.com/the-top-100-java-libraries-in-2016-after-analyzing-47251-dependencies/

IKEA Apps [online], IKEA UNITED KINGDOM. 1999 - 2018 [cit. 2018-04-30]. Dostupné z: https://www.ikea.com/gb/en/customer-service/ikea-apps/

Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions) [online], 2016. Statista [cit. 2018-04-30]. Dostupné z: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

ISO/IEC 25000:2014, *Systems and software Quality Requirements and Evaluation (SQuaRE)*, 2014. 2.

IT & Security Systems [online], 2018. ATIcomp [cit. 2018-04-30]. Dostupné z: http://www.aticomp.sk/

Jak začít používat eRecept, 2017b. *Elektronická preskripce* [online]. Praha: SÚKL [cit. 2018-04-30]. Dostupné z: https://www.epreskripce.cz/jak-zacit-pouzivat-erecept

Jak získat kvalifikovaný certifikát pro elektronický podpis, 2017c. *Elektronická preskripce* [online]. Praha: SÚKL [cit. 2018-04-30]. Dostupné z: https://www.epreskripce.cz/jak-ziskat-kvalifikovany-certifikat-pro-elektronicky-podpis

JavaScript Versions [online], 2018. W3SCHOOLS. [cit. 2018-04-30]. Dostupné z: https://www.w3schools.com/js/js_versions.asp

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

KOMZÁK, Tomáš, 2013. *Řízení IT projektů pro úplné začátečníky*. Brno: Computer Press. Pro úplné začátečníky. ISBN 978-80-251-3791-8.

KOSEK, Jiří, 2000. *XML pro každého: podrobný průvodce*. Praha: Grada. ISBN 80-716-9860-1.

KOVÁŘ, Vladimír, David KIMR, Marek ŠŤASTNÝ a Petr CIOCHOŇ, LÍZNEROVÁ, Zuzana a Lucie JANDOVÁ, ed., 2018a. *Úvod do Unicorn Application Framework*. Praha: Unicorn. ISBN 978-80-87349-22-9.

KOVÁŘ, Vladimír, David KIMR, Marek ŠŤASTNÝ a Petr CIOCHOŇ, LÍZNEROVÁ, Zuzana a Lucie JANDOVÁ, ed., 2018b. *UuApp Framework*. Praha: Unicorn. ISBN 978-80-87349-23-6.

Legislativa, 2017d. *Elektronická preskripce* [online]. Praha: SÚKL [cit. 2018-04-30]. Dostupné z: https://www.epreskripce.cz/legislativa

LELLA, Adam, 2015. *Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S.* [online]. COMSCORE, INC. [cit. 2018-04-29]. Dostupné z: https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S

LEWIN, Erland, 1998. *The Broker Architecture* [online]. Center for Speech Technology [cit. 2018-04-30]. Dostupné z: https://www.speech.kth.se/broker/

MALLAWAARACHCHI, Vijini, 2017. *10 Common Software Architectural Patterns in a nutshell* [online]. [cit. 2018-04-30]. Dostupné z: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

MARR, Bernard, 2018a. *20 Mind-Boggling Facts About Quantum Computing Everyone Should Read* [online]. Forbes Media LLC. [cit. 2018-04-30]. Dostupné z: https://www.forbes.com/sites/bernardmarr/2018/02/23/20-mind-boggling-facts-about-quantum-computing-everyone-should-read/#4872ae615edb

MARR, Bernard, 2018b. *The Key Definitions Of Artificial Intelligence (AI) That Explain Its Importance* [online]. Forbes Media LLC. [cit. 2018-04-30]. Dostupné z: https://www.forbes.com/sites/bernardmarr/2018/02/14/the-key-definitions-of-artificial-intelligence-ai-that-explain-its-importance/#42104b84f5d8

MCGINNIS, Tyler, 2016. *Imperative vs Declarative Programming* [online]. [cit. 2018-04-30]. Dostupné z: https://tylermcginnis.com/imperative-vs-declarative-programming/

MĚŘINSKÝ, Jan, Milan BRAŠNA a Kateřina ŠTROMEROVÁ, 2018. *Dokumentace +4U eRecept*. Praha: Unicorn. ISBN 978-80-87349-68-7.

MORROW, Kim, 2014. Web 2.0, Web 3.0, and the Internet of Things. *UX Booth* [online]. [cit. 2018-04-30]. Dostupné z: http://www.uxbooth.com/articles/web-2-0-web-3-0-and-the-internet-of-things/

MUJUMDAR, Ashwini, Gayatri MASIWAL a P. M. CHAWAN, 2012. Analysis of various Software Process Models. *International Journal of Engineering Research and Applications*. 2012(3), 3-7. ISSN 2248-9622. Dostupné také z: https://www.researchgate.net/publication/316510707_Analysis_of_various_Software_Process_Models

NÝVLT, Václav, 2017. Největší problém umělé inteligence: nelze zjistit, „proč“ se rozhodla. *IDNES.cz* [online]. [cit. 2018-04-30]. Dostupné z: https://technet.idnes.cz/umela-inteligence-problem-rozhodovani-dye-/tec_technika.aspx?c=A170619_094905_tec_technika_nyv

O receptu, 2017e. *Elektronická preskripce* [online]. Praha: SÚKL [cit. 2018-04-30]. Dostupné z: https://www.epreskripce.cz/e-recept

PALKOVIČ, Lukáš, 2017. SEO 2018: Co nás čeká a nemine?. *EVisions Advertising* [online]. [cit. 2018-04-30]. Dostupné z: https://www.evisions.cz/blog-2017-12-14-seo-2018-co-nas-ceka-a-nemine/

PANEC, Zdeněk, 2003. *IT Systems* [online]. CCB, **2003**(6) [cit. 2018-04-30]. Dostupné z: https://www.systemonline.cz/clanky/co-je-to-business-intelligence.htm

PATEL, Parikshit, 2014. *Three Layer Architecture in C# .NET* [online]. CodeProject [cit. 2018-04-30]. Dostupné z: https://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET

PAVLÍČEK, Antonín a Alexander GALBA, 2012. *Moderní informatika*. Praha: Professional Publishing. ISBN 978-80-7431-109-3.

PAWLAN, Monica, 1999. *Essentials of the Java Programming Language, Part 1* [online]. ORACLE. [cit. 2018-04-30]. Dostupné z: http://www.oracle.com/technetwork/java/index-138747.html

POTÁČEK, Jiří. Informační a komunikační technologie. In: *KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2014 [cit. 2018-04-22]. Dostupné z: http://aleph.nkp.cz/F/?func=direct&doc_number=000000026&local_base=KTD

POŽÁR, Josef. Vybrané hrozby informační bezpečnosti organizace. *CyberSecurity.cz* [online]. 2017 [cit. 2018-04-29]. Dostupné z: https://www.cybersecurity.cz/data/Pozar2.pdf

Progressive Web Apps [online], 2018. GOOGLE. [cit. 2018-04-30]. Dostupné z: https://developers.google.com/web/progressive-web-apps/

PROCHÁZKA, Michal, 2018. *Data mining: jiný pohled na problém* [online]. [cit. 2018-04-30]. Dostupné z: http://vtm.e15.cz/aktuality/data-mining-jiny-pohled-na-problem

PUTANO, Ben, 2017. *Most Popular and Influential Programming Languages of 2018* [online]. Stackify [cit. 2018-04-30]. Dostupné z: https://stackify.com/popular-programming-languages-2018/

REWALLEN, Raymond, 2005. 4 major principles of Object-Oriented Programming. *CodeBetter.com* [online]. [cit. 2018-04-30]. Dostupné z: http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/

RIGHTSCALE, INC., *State of the Cloud Report: Public Cloud Adoption Grows as Private Cloud Wanes*, 2017. Santa Barbara. Dostupné z:

https://assets.rightscale.com/uploads/pdfs/RightScale-2017-State-of-the-Cloud-Report.pdf

ROBERTS, Mike, 2016. *Serverless Architectures* [online]. [cit. 2018-04-30]. Dostupné z: https://martinfowler.com/articles/serverless.html

ROSIC, Ameer, 2017. l. *What is Cryptocurrency: Everything You Need To Know [Ultimate Guide]* [online]. [cit. 2018-04-30]. Dostupné z: https://blockgeeks.com/guides/what-is-cryptocurrency/

ROUDENSKÝ, Petr, 2016. *Kvalita softwaru: teorie a praxe*. Prostějov: Computer Media. ISBN 978-80-7402-294-4.

Sekvenční diagramy UML: Referenční dokumentace [online], 2018b. MICROSOFT. [cit. 2018-04-30]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/dd409377.aspx

SHARKIE, Craig a Andrew FISHER, 2015. *Responzivní webdesign: okamžitě*. Brno: Computer Press. ISBN 978-80-251-4384-1.

SHUJA, Ahmad K. a Jochen. KREBS, c2008. *IBM Rational Unified Process reference and certification guide: solution designer*. Upper Saddle River, NJ: IBM Press/Pearson. ISBN 978-0-13-156292-9.

SHUKLA, Shashvat, 2017. Framework vs Library vs Platform vs API vs SDK vs Toolkits vs IDE. *Medium* [online]. [cit. 2018-04-30]. Dostupné z: https://medium.com/@shashvatshukla/framework-vs-library-vs-platform-vs-api-vs-sdk-vs-toolkits-vs-ide-50a9473999db

SOAP vs. REST: The Differences and Benefits Between the Two Widely-Used Web Service Communication Protocols [online], 2017. STACKIFY. [cit. 2018-04-30]. Dostupné z: https://stackify.com/soap-vs-rest/

SOMMERVILLE, Ian, 2013. *Softwarové inženýrství*. Brno: Computer Press. ISBN 978-80-251-3826-7.

Top 21 Software Development Tools in 2018 [online], 2018. GURU99. [cit. 2018-04-30]. Dostupné z: https://www.guru99.com/software-development-tools.html

V jaké podobě může lékař pacientovi eRecept předat?, 2017f. *Elektronická preskripce* [online]. Praha: SÚKL [cit. 2018-04-30]. Dostupné z: https://www.epreskripce.cz/v-jake-podobe-muze-lekar-pacientovi-erecept-predat

VACH, Martin, 2018. *Historie biometrik a jejich využití ve výpočetní technice* [online]. MASARYKOVA UNIVERZITA. [cit. 2018-04-30]. Dostupné z: https://www.fi.muni.cz/usr/jkucera/pv109/2003/xvach_biometriky.htm

VELTE, Anthony T., Toby J. VELTE a Robert C. ELSENPETER, 2011. *Cloud Computing: praktický průvodce*. Brno: Computer Press. ISBN 978-80-251-3333-0.

VLACH, Miroslav, 2008. *Software* [online]. [cit. 2018-04-29]. Dostupné z: http://www.mira-vlach.cz/software-definice

VLACH, Miroslav, 2012. *Jasně rozdělte role a odpovědnost* [online]. [cit. 2018-04-29]. Dostupné z: http://www.mira-vlach.cz/rozdeleni-roli-a-odpovednosti-v-projektech

VOŘÍŠEK, Lukáš, 2017. *Když virtuální realita nestačí aneb co je to Windows Mixed Reality?* [online]. INSMART. [cit. 2018-04-30]. Dostupné z: https://insmart.cz/kdyz-virtualni-realita-nestaci-aneb-windows-mixed-reality/

VR? AR? MR? Sorry, I'm confused. [online], 2018. THE FOUNDRY. London [cit. 2018-04-30]. Dostupné z: https://www.foundry.com/industries/virtual-reality/vr-mr-ar-confused

WASSERMAN, Anthony I., 2011. How the Internet transformed the software industry. *Journal of Internet Services and Applications*. **2**(1), 11-22. DOI: 10.1007/s13174-011-0019-x. ISSN 1867-4828. Dostupné také z: http://www.springerlink.com/index/10.1007/s13174-011-0019-x

Web Authentication: An API for accessing Public Key Credentials Level [online], 2018. W3C CANDIDATE RECOMMENDATION. [cit. 2018-04-30]. Dostupné z: https://www.w3.org/TR/webauthn/

Web, Hybrid, or Native Apps? What's the difference? [online], 2018. MOBILOUD. [cit. 2018-04-30]. Dostupné z: https://www.mobiloud.com/blog/native-web-or-hybrid-apps/

What is UML [online], 2018. OBJECT MANAGEMENT GROUP [OMG], INC. [cit. 2018-04-30]. Dostupné z: http://www.uml.org/what-is-uml.htm

WIEGERS, Karl Eugene, 2008. *Požadavky na software*. Brno: Computer Press. ISBN 978-80-251-1877-1.

YOSHIDA, Hu, 2018. *Top 10 IT Trends, Part 4: Biometric Authentication, Agile Methodology, And Co-Innovation*[online]. [cit. 2018-04-30]. Dostupné z: http://www.digitalistmag.com/cio-knowledge/2018/02/28/top-10-it-trends-part-4-biometric-authentication-agile-methodology-co-innovation-05809076

Začínáte s User Experience? [online], 2018. ASOCIACE UX, Z. S. [cit. 2018-04-30]. Dostupné z: http://www.asociaceux.cz/zacinate-s-user-experience

ŽÁRA, Ondřej, 2015. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press. ISBN 978-80-251-4573-9.

ŽOLTÁ, Lucie, 2018. *Typy testů a V-model* [online]. [cit. 2018-04-30]. Dostupné z: http://lucie.zolta.cz/index.php/statnice-vsb/167-typy-testu-a-v-model

ŽŮREK, Michal, 2018. *1. díl - Úvod do objektově orientovaného programování ve VB.NET* [online]. [cit. 2018-04-30]. Dostupné z: https://www.itnetwork.cz/vbnet/oop/vbnet-tutorial-uvod-do-objektove-orientovaneho-programovani