ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA EKONOMICKÁ

Diplomová práce

Analýza nákladů při support a maintenance

Cost analysis for support and maintenance

Bc. Tomáš Vyleta

Plzeň 2023

Čestné prohlášení

Prohlašuji, že jsem diplomovou práci na téma

„Analýza nákladů při support a maintenance“

vypracoval samostatně pod odborným dohledem vedoucího diplomové práce **doc. RNDr. Mikuláše Gangura, Ph.D.** za použití pramenů uvedených v přiložené bibliografii.

Plzeň dne 23. 4. 2023 v. r. *Bc. Tomáš Vyleta*

Zásady pro vypracování práce

1. Charakterizujte oblast řízení nákladů na support a maintenance.
2. Představte složení nákladů oddělení supportu a maintenance a jejich členění.
3. Představte současné metody predikce nákladů a vybrané popište.
4. Proveďte sběr dat o současných odhadech a nákladech, popište je.
5. Analyzujte a vyhodnoťte data podle zvolených ukazatelů a přesnosti odhadu, aplikujte vybrané metody predikce.
6. Vytvořte aplikaci pro predikci a vizualizaci dat.

Poděkování

Nejprve bych chtěl poděkovat smému vedoucímu práce **doc. RNDr. Mikuláši Gangurovi, Ph.D.**, který mi poskytl mnoho užitečných rad a připomínek a věnoval mi svůj čas a energii při konzultacích. Z řad společnosti Diebold Nixdorf je to pak **Ing. David Krivánka**, který mi poskytl cenné informace, materiály a hlavně svůj čas pro řešení této práce. Dále bych chtěl poděkovat **Mgr. Evě Konířové** za pravopisnou a interpunkční opravu práce. Nakonec bych rád poděkoval své rodině a přátelům, kteří mě podporovali po celou dobu psaní práce.

Obsah

Úvod

Vývoj softwaru je zdrojově nákladná činnost jak z hlediska lidského kapitálu, tak i časového a finančního. Cílem vývoje je uspokojit zákazníkovy požadavky, které se časem mohou měnit, ať už je to zapříčiněno změnou prostředí, trendů nebo požadavků na software. Životní cyklus softwaru nekončí okamžikem, kdy je distribuován zákazníkovi, ale je potřeba jej neustále inovovat, aby uspokojil požadavky, které jsou na něho kladeny. Pokud by nebyl spolehlivý a užitečný, uživatelé by ho přestali používat a v dnešní době, kdy na různých systémech jsou postaveny ekonomické burzy, zdravotní systémy atd., nefunkčnost či chybovost nepřipadá v úvahu, proto je nutné dbát na jeho údržbu (anglicky *Maintenance*) a podporu (anglicky *Support*). Údržba je důležitou fází životního cyklu softwaru a napomáhá k opravě chyb, které nebyly odstraněny při vývoji, nebo zakomponování nových komponent.

S údržbou a podporou se ovšem vážou i náklady a ty bývají většinou větší než samotný vývoj nebo ostatní fáze životního cyklu softwaru. Tyto nezanedbatelné náklady nutí společnosti k jejich plánování a k odhadům na další období. Za tímto účelem se provádí analýza nákladů (anglicky *Cost Analysis*), která bude v kontextu se softwarovou podporou a údržbou tématem této diplomové práce — „*Analýza nákladů při support a maintenance*“. Samotná společnost **Diebold Nixdorf** nabídla toto téma ke zpracování, jelikož hledá možnosti, jak lépe predikovat či odhadovat budoucí náklady na údržbu, aby mohla efektivněji plánovat financování podniku.

Společnost působí v odvětví retail (maloobchodních) služeb se zákazníky po celém světě. Denně přicházejí od zákazníků požadavky, které je nutné řešit, jelikož narušují chod jejich služeb, jež společnost poskytuje. U daného požadavku společnost předem neví, jak dlouho bude trvat napravení problému. Avšak do společnosti se dostává požadavek již s nějakými informacemi, které jsou zaznamenávány, a analýza nákladů se bude zaměřovat právě na ně. Poskytování softwaru také s sebou přináší proces zajišťovaní kvality, efektivity a účinnosti IT služeb, které musí být nějak stanovené. K tomuto účelu společnost využívá soubor osvědčených postupů a standardů ITIL.

Obsah a cíl práce

Hlavním cílem diplomové práce je pomocí statistických metod provést analýzu nákladů na podporu a údržbu a demonstrovat predikci těchto nákladů pomocí vytvořeného programu. Dílčími podcíli jsou:

a) představit hlavní činnosti a stručnou historii společnosti Diebold Nixdorf, s.r.o.,

b) představit životní cyklus vývoje software a metody řízení,

c) popsat fázi maintenance v kontextu softwarového životního cyklu,

d) představit klíčové oblasti pro maintenance z rámce ITIL,

e) popsat support a maintenance ve společnosti Diebold Nixdorf, s.r.o.,

f) určit cíle statistického výzkumu,

g) nalézt a navrhnout vhodné metody analýzy nákladů,

h) sebrat data z firemní databáze a jejich příprava,

i) vytvořit funkční aplikaci pro analýzu nákladů,

j) na základě výsledků statistických testů provést interpretaci.

V první kapitole je představena společnost Diebold Nixdorf, s.r.o, pro kterou je téma diplomové práce zpracováváno. Je zde krátká historie společnosti, její současná situace a zmíněn je také účel plzeňské pobočky, kterou autor navštěvoval. Druhá kapitola definuje základní vymezení softwaru jako produktu poskytovatelů softwaru, dále je představen životní cyklus vývoje softwaru (SDLC) a jeho fáze, kde jednou z fází je údržba. Dále jsou v kapitole představeny vybrané metody přístupu k řízení SDLC a představen DevOps. Další kapitola navazuje na fázi údržby, zde je popsána oblast působení v rámci firmy, jednotlivé role projektového týmu údržby a jednotlivé fáze údržby. Na to navazuje rozdělení údržby do čtyř oblastí podle dostupné literatury a popis nákladů na údržbu. Poslední teoretická kapitola se zabývá rámcem ITIL, v ní jsou popsány nejlepší praktiky související s údržbou. V páté kapitole je shrnutí rešerše a provedena kritická diskuse. V šesté kapitole již začíná praktická část práce, kde je představen model podpory společnosti včetně současného modelu výpočtu predikce nákladů a faktorů, které se podílejí na odhadovaných nákladech. V následující kapitole se nachází metodika, jež popíše hlavní cíl práce, definuje výzkumné otázky a metodologii pro statistické metody. V osmé kapitole je popsán proces sběru dat a provedena explorační analýza. Následuje analýza dat v rámci které jsou testovány hypotézy a navrženy metody klasifikace celkově stráveného času na incidentu (úsilí). V desáté kapitole je popsán vývoj aplikace. Poslední kapitola interpretuje výsledky z praktické části a odpovídá na určené výzkumné otázky.

1 Představení společnosti

Tato diplomová práce je zpracovávána pro společnost Diebold Nixdorf s.r.o. (dále pouze DN), proto tato kapitola bude věnována představení této společnosti a jejich plzeňské pobočky.

1.1 Základní informace

Jedná se o společnost s ručením omezením, která se zapsala do českého obchodního rejstříku 30. července 1999 s hlavním sídlem na adrese *Siemensova 2716/2, Stodůlky, 155 00 Praha*. Jednateli společnosti jsou Stanislav Zrcek a Vladimír Drda. Úplným společníkem je **WINCOR NIXDORF International, GmbH**, s vkladem 37 mil. Kč. Předmětem podnikání DN je výroba, instalace, opravy elektrických strojů a přístrojů, elektronických a telekomunikačních zařízení a výroba, obchod a služby neuvedené v přílohách 1 až 3 živnostenského zákona. Jedná se zejména o výrobu a dodávání informačních systémů pro oblast bankovnictví a maloobchodu (anglicky *Retail*) u kterého nabízí také servis a poradenské služby, mezi nimiž zaujímá přední příčky na trhu. (Kurzy.cz, 2023)

Obrázek 1: Logo Diebold Nixdorf, s.r.o.

Zdroj: (Wikipedie, 2023a)

1.2 Historie Diebold

Společnost s původním názvem **Diebold Bahmann Safe Company** byla založena v roce 1859 v Cincinnati (USA, stát Ohio) Charlesem Dieboldem jako společnost, která se zabývala výrobou sejfů a trezorů pro bankovní instituce. V roce 1881 dostala zakázku na první zahraniční objednávku a do roku 1936 pomalu expandovala výrobu a rozšířila ji i o výrobu pancířů pro tanky, které vyráběla i během 2. světové války. V roce 1959 měla obrat 1,7 mil. dolarů a o pět let později vstoupila na newyorskou burzu (NYSE) pod zkratkou **DBD**. V 70. letech začala

společnost nabízet CCTV systémy a díky vedení společnosti se zájem upínal také na trh s ATM (bankomaty). V roce 1973 vyvinula svůj první vlastní bankomat TABS 500 a v roce 1989 již měla 12% podíl na světovém trhu bankomatů. Na přelomu tisíciletí společnost představila první detektor sítnice na světě a také automatický, mluvící bankomat. V roce 2003 měla již obrat 2,1 mld. dolarů. Později, v roce 2008, se společnost začala zaměřovat na mobilní bankovnictví a na tři roky se stala největším výrobcem bankomatů v USA. Rok 2015 byl pro společnost opravdu zlomový, jelikož představila dva nové, průlomové, bankomaty. „Irving“ umožňoval vybírat si hotovost pomocí skeneru sítnice namísto karty a „Janus“ umožňoval obsloužení dvou zákazníků najednou. Ve stejném roce, přesněji 23. listopadu 2015, oznámila společnost akvizici s německou společností **Wincor Nixdorf** čímž ovládly 35 % světového trhu s bankomaty. 24. května 2016 byla akvizice dokončena, proběhla fúze společností a zrodila se společnost **Diebold Nixdorf** tak, jak ji známe v dnešní podobě. (Wikipedie, 2023a)

1.3 Historie Wincor Nixdorf

Společnost s původním názvem **Nixdorf Computer, AG**, byla založena v roce 1952 v Paderbornu (Německo, okres Paderborn) Heinzem Nixdorfem. Zabývala se výrobou počítačů, přesněji kalkulaček a jako první na světě představila kalkulačku se zabudovanou tiskárnou. V roce 1968 se stalo trendem elektronické zpracování dat, proto společnost rozhodla o výrobě takového počítače, který by si mohly pořídit i rodiny do domácnosti. V 70. letech společnost rostla a expandovala do dalších zemí, až se stala 4. největším výrobcem počítačů v Evropě. V roce 1985 měla společnost obrat 4 mld. německých marek a na 23 000 zaměstnanců ve 44 zemích. 1. října 1990 kupuje Nixdorf společnost Siemens a vzniká dceřiná společnost Siemens Nixdorf Informationssysteme (SNI), která se stává největší počítačovou společností v Evropě. Opět 1. října, avšak roku 1999 přechází pod jiné vlastnictví a přejmenovává se na **Wincor Nixdorf** a zabývá se výrobou bankomatů, pokladen a vratných automatů na láhve. V roce 2016 se slučuje se společností Diebold a vzniká DN. (Wikipedie, 2023b)

1.4 Současnost

V současné době je společnost rozdělena na tři regionální divize působící ve 130 zemích světa a zaměstnává na 23 000 zaměstnanců. Generálním ředitelem je **Octavio Marquez**. Společnost je globálním producentem a dodavatelem hardwaru, softwaru a služeb v oblasti bankovnictví a maloobchodu. Momentálním trendem, kterým se společnost zabývá, je automatizace, digitalizace a transformace. (Diebold Nixdorf, 2023a)

1.5 Global Center v Plzni

V Plzni, konkrétněji v centru Avalon kanceláří, se nachází pobočka **Global Center** (dále pouze GC), což je jedno ze šesti vývojových center DN. V Plzni se pobočka zaměřuje zejména na

doménu bankovnictví. Mezi hlavní činnosti patří vývoj bankovního systému Vynamic View, který se stará o živý monitoring bankomatů. GC vzniklo v roce 2015 a v současné době zde pracuje na 150 zaměstnanců. Jsou zde přítomny 3 týmy - **Research & Development**, který se zaměřuje právě na vývoj systému Vynamic View, dále **Delivery**, který upravuje řešení zákazníkům na míru a **Maintenance & Support**, který se stará o údržbu a zkvalitnění služeb produktů. Do posledně zmiňovaného týmu spadá téma této diplomové práce. (Diebold Nixdorf, 2023b)

Společnost DN se řídí následujícím prohlášením o kvalitě: „Kvalita pro DN znamená poskytování inovativních a spolehlivých výrobků a plnění našich závazků vůči zákazníkům, zaměstnancům a dodavatelům. Neustálé zlepšování je základem našeho trvalého úspěchu.“[1]. (Diebold Nixdorf, 2023c)

[1] Vlastní překlad autora diplomové práce.

2 Vývoj softwaru

V následujících kapitolách budou představeny teoretické základy diplomové práce. Autor představí základní pojmy včetně definice softwaru a jeho životního cyklu až po **softwarovou podporu**, v anglickém překladě *Software Support*, v rámci které se provádí **softwarová údržba** (anglicky *Software Maintenance*). V praktické části této práce provede autor z nákladů (z oblasti softwarového životního cyklu) **analýzu nákladů** (anglicky *Cost Analysis*).

2.1 Definice pojmů

Ještě než se autor bude zabývat vývojem softwaru a jeho údržby, je nutné definovat základní pojmy, které se objevují v celé oblasti životního cyklu softwaru. Na začátek je nutné definovat rozdíly mezi službou a produktem za účelem definice **softwaru**, o kterým se boudou zabývat následující kapitoly.

2.1.1 Software

Služba je nehmotný soubor výhod nebo činností, jež prodává poskytovatel zákazníkovi, který z ní přijímá užitek. (Management Mania, 2016)

Produkt je vše, co lze nabídnout zákazníkovi ke spotřebě za cílem uspokojení jeho potřeb. Je výsledkem činnosti organizace a může mít hmotnou, či nehmotnou podobu. (Management Mania, 2015)

Software je tedy nehmotná podoba produktu, který je nabízen zákazníkům většinou jako autorské dílo v podobě licencí. (Wikipedie, 2023c) Software se skládá nejen z kódu, ale patří k němu také dokumentace, specifikace, design, uživatelské příručky a procedury ke spuštění a ovládání. (Grubb & Takang, 2007, s. 7)

Mezi hlavní rozdíly produktu a služby patří:

- **Nehmotnost**
  Služba se skládá z výhod a činností, které jsou nehmatatelné (nemohou být viděny, cítěny atd.), nelze je inventarizovat a patentovat[2] na rozdíl od produktu. Také samotné stanovení ceny je obtížnější než u produktu.

- **Heterogenita**
  Jelikož služba se skládá z činností a ty jsou vykonávány lidmi, jsou služby více heterogenní než produkty, jelikož lidská práce je závislá na několika faktorech, které se dají těžko řídit. Jedná se například o ochotu zákazníka sdělit své požadavky nebo ochotu

[2]Lze je však chránit autorskými právy, nebo ochrannými známkami, popř. licencemi.

zaměstnance vyhovět těmto požadavkům, což dělá výsledek velmi obtížným, pokud se má služba dodat podle nějakého plánu nebo specifikace.

- **Současná výroba a spotřeba**
  Služby jsou produkovány a spotřebovávány zároveň na rozdíl od produktu, kdy lze jeho spotřebu rozdělit.

- **Pomíjivost**
  Služba existuje pouze, když je vykonávána, nelze ji uložit ani skladovat, není možné ji ani vrátit nebo přeprodat jiné straně. (Niessink & van Vliet, 2000, s. 104-105)

Občas je služba dodávána s produktem a tvoří jeden celek. Příkladem je společnost DN, která dodává hardware + software (produkt) a k tomu je nedílnou součástí i servis a údržba (služba). To znamená, že kvalita bude souzena podle obou kritérií. Výstup vývoje softwaru je produkt a u softwarové podpory je produktem služba. (Niessink & van Vliet, 2000, s. 105)

2.1.2 Kvalita

Při vývoji je jeden z cílů dosahování kvality, ale co je kvalita v kontextu vývoje softwaru? Podle Hudec a kol. (2012, s. 58) je kvalita: „Schopnost produktu, služby nebo procesu poskytovat zamýšlenou hodnotu.“ Pokud software nebo služba funkcionálně pracuje podle požadavků a dá se na tyto funkcionality spolehnout, jedná se o vysokou kvalitu. V dlouhodobém časovém měřítku se jedná také o zvýšení efektivnosti a zlepšení hospodárnosti (šetření nákladů). Zákazník má právo definovat si úrovně kvality softwaru na poskytovateli, jelikož je ten, který za to platí. (Hudec a kol., 2012, s. 58; Hüttermann, 2012, s. 51)

2.1.3 Zákazník

Zákazník je jedna z rolí zainteresovaných stran. Zákazníkem je osoba, nebo skupina, která definuje požadavky na daný software nebo službu, kupuje ji a přebírá ji do svého užití. Je také zodpovědný za její výsledky. (Axelos, 2019, s. 23)

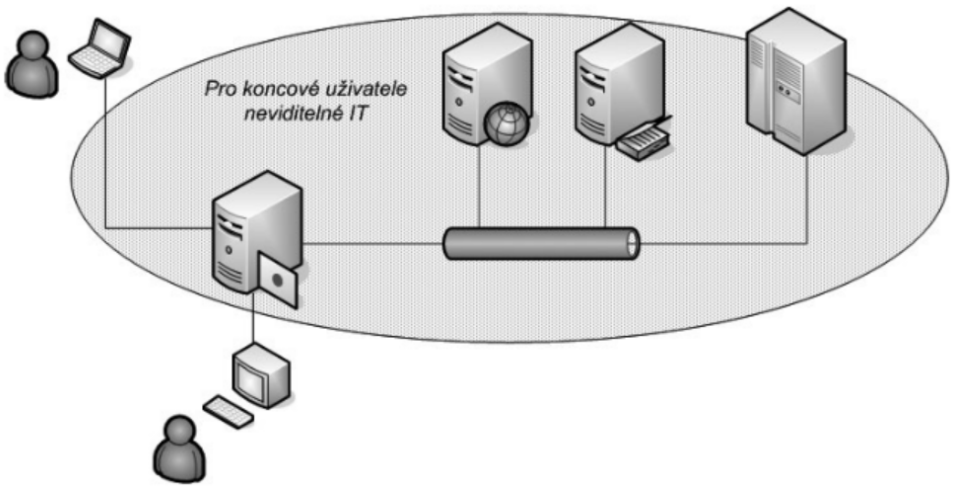
Zákazníky dělíme do dvou skupin, a to na zákazníky **interní**, kteří jsou součástí stejné organizace, a na zákazníky **externí**, kteří nejsou součástí organizace. (Hudec a kol., 2012, s. 33-39)

2.1.4 Poskytovatel

Poskytovatel je osoba, nebo skupina, která poskytuje a vytváří daný software nebo službu zákazníkovi, je tedy jeho tvůrcem. Vztahy poskytovatele a zákazníka mohou být různé, jak již bylo zmíněno, poskytovatel může nabízet software/službu uvnitř společnosti, kde zákazníkem bude interní oddělení, poskytovat je jiným společnostem, nebo nabízet na trhu, kde je mohou ostatní zákaznici poskytovat dále. (Axelos, 2019, s. 22)

Z výše uvedených definicí můžeme stanovit pojem vycházející z pojmu služby, a to **IT služba**, čímž se rozumí poskytování funkčnosti softwaru koncovým uživatelům za účelem zefektivnění jejich podnikových procesů. Zdroj Procházka & Klymeš (2011, s. 21) ji definuje následovně: „Služba je prostředek dodávání hodnoty zákazníkovi tím, že zprostředkovává výstupy, jichž chce zákazník dosáhnout, aniž by vlastnil specifické náklady nebo rizika.“. Jedná se tedy o propojení IT (= *Information Technology*) a služeb. Jde o různé propojení komponent hardwaru (server, síťové prvky atd.) a softwaru (databáze, firewally, operační systémy atd.), které koncovému uživateli nejsou viditelné a jakoukoli chybu či pokles kvality by měl ihned reportovat poskytovateli služby. Když autor zmíní službu v této diplomové práci, bude myslet právě IT službu. Pro představu je uveden obrázek 2, který ukazuje, co zákazník skutečně využívá a co pro něho není viditelné.

Obrázek 2: Šíře IT služby



Zdroj: (Procházka & Klymeš, 2011, s. 22)

2.1.5 Dohoda o úrovni služeb

Cíle a kvalita poskytovaných IT služeb včetně odpovědnosti jednotlivých stran, způsobu měření a reportování závad jsou definovány v dokumentu zvaném **Dohoda o úrovni služeb** (SLA = *Service Level Agreement*). (Procházka & Klymeš, 2011, s. 22) Pokud nejsou podmínky SLA dodržovány, může zákazník požadovat od poskytovatele odškodnění či náhradu škod za narušení chodu jeho podnikání. Stanovení SLA zaručuje, že se v případě problémů nebude moci ani jedna strana „vymlouvat“ na chování druhé strany. Na straně poskytovatele to také vede k odpovědnosti a produktivitě tím, že se udržuje definovaná efektivita a nebudí se hromadí se problémy, což zabránění zákazníkovi odejít a zvýší jeho spokojenost z využívání služeb. (LiveAgent, 2022a)

Nyní jsou představeny všechny pojmy potřebné k definici softwarové podpory a softwarové údržby, které jsou vysvětleny dále.

2.1.6 Softwarová podpora (*Software Support*)

Softwarová podpora je podle zdroje Procházka & Klymeš (2011, s. 31) definována jako: „Podporou produktu rozumíme poskytování informací, pomoci a školení za účelem instalace či provozu softwaru v prostředí k tomu určeném a také jí rozumíme distribuci zlepšených funkčností softwaru ke koncovým uživatelům.“. Jedná se tedy o službu nabízenou poskytovatelem softwaru, která je poskytována po prvním dodání softwaru. Je to kontaktní místo mezi zákazníkem a poskytovatelem softwaru. Poskytuje technickou pomoc podle aktuálních potřeb a požadavků zákazníka. Měla by být dostupná 24/7, ačkoliv odezva se liší podle závažnosti problému stanoveného ve smlouvě, kterou strany mezi sebou mají. (Spinnaker Support, 2019; LiveAgent, 2022b)

2.1.7 Softwarová údržba (*Software Maintenance*)

Softwarová údržba podle zdroje Procházka & Klymeš (2011, s. 30) je definována jako: „Údržba softwaru je procesem modifikace softwarového systému či komponenty po doručení uživateli za účelem korekce chyby, zlepšení výkonu či jiného atributu nebo adaptace systému na změny prostředí.“. Softwarovou údržbou se bude více zabývat autor v kapitole Údržba.

Softwarová podpora a její údržba je často spojována do jednoho spojení ”*Support and Maintenance*”. Ačkoliv jsou tyto pojmy zaměňovány, lze tyto přístupy odlišit zejména podle **naléhavosti jejich přístupu**. U softwarové podpory se jedná zejména o opravu chybných částí softwaru reaktivní přístupem, na druhou stranu softwarová údržba využívá pro-aktivní přístup k přidávání nových funkcí nebo odstraňování chyb s nižší prioritou, které nenarušují chod softwaru. (Singlemind, 2022)

2.2 Životní cyklus softwaru

Životní cyklus softwaru, SDLC (= *Software Development Life Cycle*), je proces od koncepce až po konečný stav softwaru, který má za cíl vznik kvalitního a finančně udržitelného softwaru. Životní cyklus softwaru však nekončí jeho dodáním, ale je potřeba ho neustále vylepšovat a dolaďovat, aby se setkal s plánovanými požadavky a aby uspokojil měnící se požadavky nejen zákazníka, ale také uživatele, který nemusí být vždy zákazníkem. Pokud by nebyl software užitečný a spolehlivý v oblasti, pro kterou byl určen, zákazníci by ho přestali využívat. SDLC zajistí správné techniky a strategie pro dlouhodobé využívání softwaru. Vývoj softwaru je pro společnosti finančně náročný zejména kvůli průzkumu, vývoji, marketingu atd., kdyby cyklus skončil jeho vydáním, nebylo by to pro žádnou stranu z dlouhodobého hlediska přínosné. (Thales Group, 2022)

2.2.1 Historie

Cyklus vznikl v 60. letech 20. století a podle Elliott (2004) vznikl právě z potřeby velkých obchodních konglomerátů vyvíjet funkční a rozsáhlé obchodní systémy. Časem si i další výrobci hardwaru a softwaru začali tento způsob vývoje osvojovat.

2.2.2 Fáze SDLC

Proces životního cyklu je rozdělen do jednotlivých fází, které se vztahují na různou konfiguraci jak hardwaru, tak softwaru pro dosažení vysoce kvalitního celku, jenž splňuje požadavky zákazníka a požadovanou funkčnost. Fáze SDLC se skládá celkem z 10 fází, které jsou jednotlivě popsány dále:

1. **Fáze proveditelnosti (anglicky *Planning Stage*)**
   Tato první fáze, nazývaná občas také jako fáze plánování, má za účel porozumění a naplánování nového softwaru. Pomáhá k pochopení cílů, rozsahu a definování funkcí softwaru. V této fázi se také dají snadno odhalit rizika a nedostatky ještě dříve, než se začne s vývojovou fází. Také pomáhá najít a zajistit financování projektu. Fáze má mnoho společného s plánováním běžného projektu, a proto je její součástí i návrh časového plánu, finančního plánu a plánu rizik.

2. **Shromáždění požadavků (anglicky *Requirement Gatherings*)**
   Je vykonáno týmovými seniory, kteří sbírají podklady od všech stakeholderů, ale také od expertů z dané oblasti zaměření softwaru. Požadavky se následně analyzují a organizují do strukturovaného formátu, který je srozumitelný pro celý tým. Tento proces je zásadní pro úspěch projektu, protože správně definované požadavky umožňují týmu efektivněji a přesněji plánovat, navrhovat a vytvářet softwarové řešení.

3. **Analýza (anglicky *Analysis*)**
   Fáze analýzy zahrnuje sběr všech možných podkladů z minulého kroku, jako jsou požadavky na nový systém, a také vznikají nápady pro první prototypy. K prvotnímu prototypu jsou pak také připraveny jednotlivé alternativy. Výstupem této fáze je pak ujasnění specifikací na software, hardware a síťové propojení na systém, aby vyhovovaly podmínkám zákazníka a sepsaní SRS (= *Software Requirements Specification*) dokumentu. Běžně se také provádí analýza koncových uživatelů.

4. **Návrh (anglicky *Design*)**
   Fáze návrhu je klíčovou pro fázi vývoje. V této fázi je vybraný prototyp zkoumán více do hloubky a jsou ujasňovány specifikace spolu s návrhem uživatelského rozhraní (UI = *User Interface*), systémového rozhraní, datové vrstvy, návrhu sítě atd. SRS dokument z minulé fáze je více rozepsán a logicky upraven do výsledné podoby. Z projektového hlediska jsou již navrženy jednotlivé aktivity s činnostmi a milníky pro každou následující fázi SDLC.

5. **Vývoj (anglicky *Coding*)**
   Ve vývojové fázi vzniká kód aplikace a struktura podle designu a SRS dokumentu vycházejícího z minulé fáze. Vývojáři postupují podle pokynů definovaných organizací. Vývojáři musí být schopni efektivně a přesně převést požadavky a návrhy do funkčního kódu, který bude sloužit jako základ softwaru.

6. **Testování (anglicky *Testing*)**
   Vývojovou fází to nekončí, výsledný software musí být otestován, aby splňoval předem stanovená kritéria nebo legislativu a také, aby neobsahoval nějaké chyby, v programátorské hantýrce *bugy*, které by negativně ovlivnily výslednou funkcionalitu. V dokumentu SRS jsou také definovaná kvalitativní měřítka a ta je nutné rovněž splnit.

7. **Implementace a integrace (anglicky *Implementation and Integration*)**
   Po otestování vyvinutého softwaru je nutné jednotlivé separátní moduly vložit do jednoho celku, k tomu slouží fáze implementace — vzniká jedno integrované prostředí.

8. **Nasazení (anglicky *Deployment*)**
   Jakmile je software otestován, přichází na řadu fáze nasazení, kdy je otestovaný software nasazen na produkční prostředí a zákazník poprvé přichází do styku s vyvíjeným softwarem. Přicházejí na řadu UAT (= *User Acceptance Tests*) a přichází první zpětná vazba od zákazníka, který probírá zkušenosti z aplikace s vývojáři. Po UAT přichází na řadu již ostré nasazení a plný chod u zákazníka.

9. **Údržba (anglicky *Maintenance*)**
   Předposlední fází SDLC je pak fáze údržby, do které spadá i téma této diplomové práce. Po předání softwaru zákazníkovi cyklus nekončí, je nutné reagovat na zpětnou vazbu zákazníka. Zákazník může chtít nové funkcionality nebo nějaké aktualizovat na základě měnící se situace na trhu nebo legislativy. Software také může obsahovat chyby, které se nedokázaly v předešlých fázích podchytit, u nich je nutné zajisti jejich nápravu.

10. **Likvidace (anglicky *Disposal*)**
    O fázi likvidace se uvažuje jenom v tom případě, pokud uvažujeme o ukončení používání softwaru. Jsou zde vypracovány plány pro ukončení stávajícího systému, nebo případně pro přestup na nový systém. Definici popisuje zdroj (Encyclopedia, 2021) takto: „Účelem je zde řádně přesouvat, archivovat, vyřazovat nebo ničit informace, hardware a software, které mají být nahrazeny, způsobem, který zabrání jakékoli možnosti neoprávněného vyzrazení citlivých dat.“. Součástí této fáze může být také migrace, což je přesun dosavadní softwarové architektury do nové nebo aktualizované verze. (Preston, 2021; Martin, 2023; Encyclopedia, 2021; Software Testing Help, 2023)

Pro provádění, či neprovádění jednotlivých fází záleží hodně na struktuře a velikosti projektu (softwaru). Fáze jsou na sebe vzájemně závislé a mohou být kombinovány nebo se mohou překrývat.

2.2.3 Metodiky SDLC

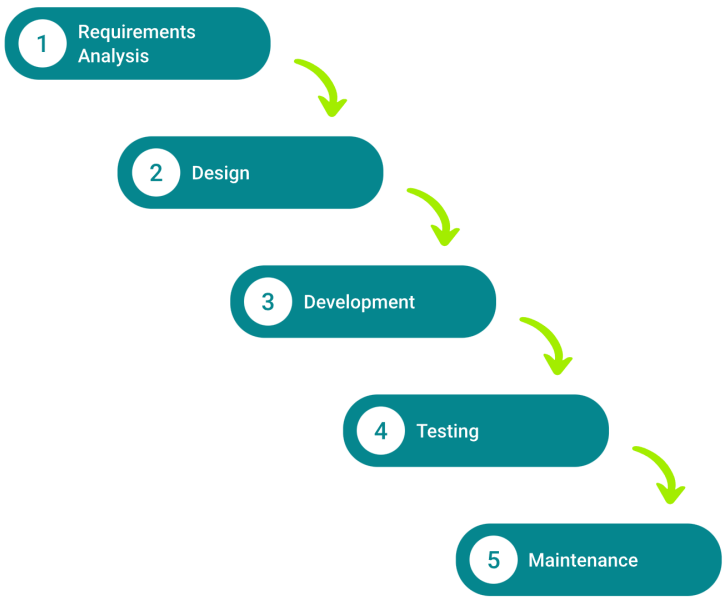
V dnešní době je běžné, že software propojuje více dalších softwarů nebo systémů do většího integrovaného celku. Tyto systémy mohou pocházet i od jiného poskytovatele z třetí strany. Ke správě a správnému zajištění požadavků softwaru se využívají metodiky, z nich některé, jsou popsány v této kapitole. Metodiky se rozdělují na **agilní metody**, které umožňují rychlé změny softwaru, **metody iterativní**, které se zaměřují na vylepšování softwaru pomocí iterací, **sekvenční modely**, které se zaměřují na plánování větších projektů a úspěšné splnění výsledků, nebo **anamorfní metody**, které se zaměřují na formu vývoje. (Encyclopedia, 2021)

V následujícím seznamu jsou představeny vybrané metody:

Metoda vodopádu (anglicky *Waterfall*)

Patří mezi sekvenční modely, kdy jednotlivé fáze jsou prováděny lineárně, to znamená, že každá fáze závisí na fázi předchozí. Tato metoda není příliš flexibilní, jelikož nabízí směr pohybu pouze dolů (anglicky *top-down*). Pokud ale u projektu není potřeba flexibilita, jedná se o velmi účinnou metodu SDLC. (Sajna, 2022)

Obrázek 3: Vodopádová metoda



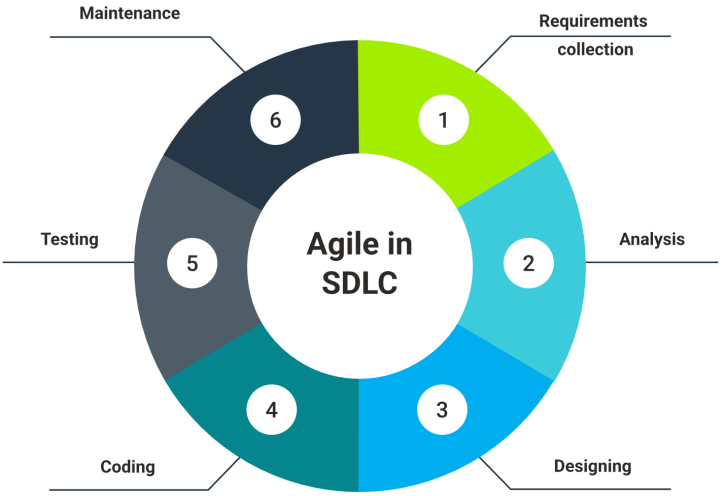
Zdroj: (Sajna, 2022)

Agilní metoda (anglicky *Agile*)

Jak autor již zmiňoval v úvodu, agilní metoda umožňuje rychlé změny ve vývoji softwaru.

Podporuje včasné dodání, neustálé zlepšování softwaru a flexibilní reakce na změny. Je založena na stále běžícím cyklu, kde každá její iterace (anglicky *sprint*) v sobě zahrnuje potřebné fáze SDLC, a tudíž software může být v rámci každého sprintu předán zákazníkovi. (Encyclopedia, 2021)

Obrázek 4: Agilní metoda

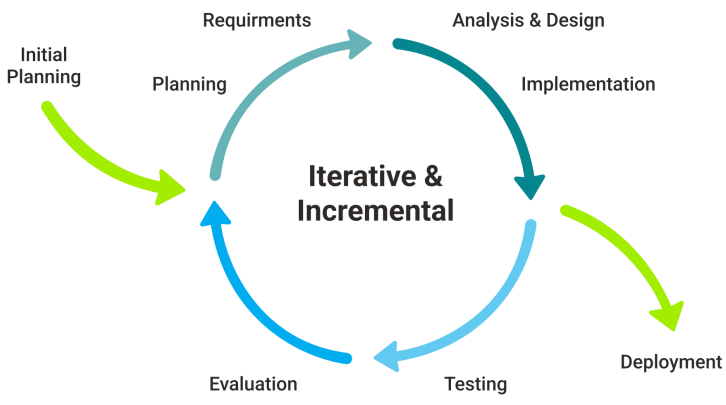


Zdroj: (Sajna, 2022)

Iterativní (přírůstkový, inkrementální) model

Jak název napovídá, jedná se o iterativní model, který je založen na postupném vylepšování pomocí přírůstků. Výsledný produkt je rozdělen na menší komponenty (anglicky *Build*), které jsou dodány zákazníkovi až po jejich dokončení. Tím se zabrání dlouhému vývoji a zavádění nového systému najednou. Prvním přírůstkem se zákazníkovi dodá základní produkt a další přírůstky vždy ten stávající vylepšují o nové funkce, dokud nejsou všechny požadavky na produkt splněny. (Encyclopedia, 2021)

Obrázek 5: Iterativní model

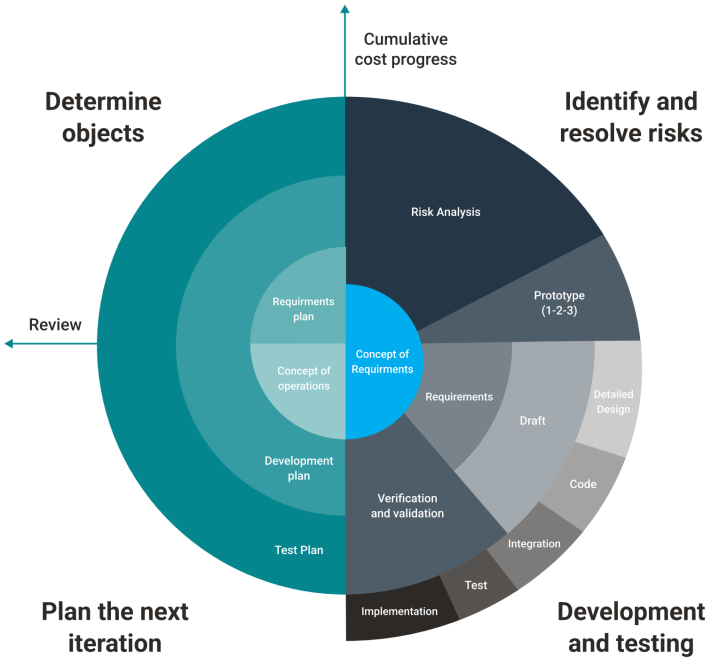


Zdroj: (Sajna, 2022)

Spirálový model

Tento model je založen na rizicích vzniklých během SDLC, model umožňuje kombinaci více přístupů a širokou adaptaci k přístupu jednotlivých specifikací kladených na software. Skládá se ze čtyř fází: plánování, analýza rizik, vývoj a vyhodnocení — které vždy zahájí novou iteraci spirály. Z uváděných metod se jedná o tu nejvíce komplexní a v měřítku rozsáhlých projektů o jednu z nejúčinnějších. (Sajna, 2022)

Obrázek 6: Spirálový model



Zdroj: (Sajna, 2022)

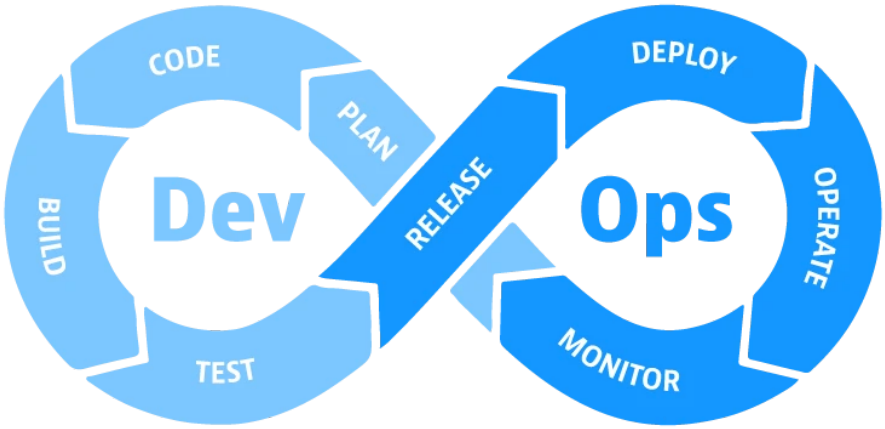
2.3 DevOps

Poslední podkapitola je DevOps, což je složenina výrazů *Development* (Dev) a *IT Operations* (Ops). Jedná se o přístup k softwaru, který propojuje vývoj softwaru a jeho provoz. Základem je spolupráce mezi těmito dvěma odděleními, na straně vývoje softwaru jsou to programátoři, testeři a zaměstnanci kontroly kvality (QA = *Quality Assurance*), na straně provozu softwaru jsou to zejména experti, kteří se starají o nasazení a chod softwaru, mezi ně patří administrátoři, správci databází a správci sítí. Přístup umožňuje sdílet stejné cíle a řešit konflikty vyvolané mezi těmito odděleními, které jsou nevyhnutelné, jelikož mají rozdílné cíle. Cílem vývojářů je **změna** — implementování nových funkcionalit a cílem provozu je eliminace rizika neboli **stabilita** softwaru např. pomocí rámce ITIL, který bude představen v kapitole ITIL. (Hüttermann, 2012, s. 4)

Podle Hüttermann (2012, s. 11) je přístup provozní části až několik let opožděn oproti vývoji softwaru, který uplatňuje agilní procesy tvorby softwaru. Mezi hlavní problémy patří zejména

rostoucí náklady na provoz a údržbu. DevOps se snaží tuto propast zmenšit pomocí spolupráce, automatizace a neustálého zlepšování. Má za cíl zkrácení celého životního cyklu softwaru a dodání co nejvíce kvalitního výsledku podle potřeb zákazníků za účelem snížení nákladů. Celý opakující se cyklus včetně jednotlivých fází SDLC představuje obrázek 7.

Obrázek 7: Smyčka DevOps



Zdroj: (Gunja, 2022)

Hlavní metodou, které DevOps využívá, jsou **malé dávky**, jež napomáhají zkrátit cyklus dodání díky menším fragmentům kódu. Funkcionalita je tímto přístupem dodávána mnohem rychleji. Malé dávky napomáhají více se zaměřit na individuální oblast a snižují potencionální riziko. Jsou lépe uchopitelné a dají se snáze testovat. Při hledání chyb je výhodou také to, že se nemusí vracet zpět o velké množství kódu, ale malé dávky umožní vrátit se k menším, lépe analyzovatelným celkům. (Hüttermann, 2012, s. 40)

Mezi výhody přístupu DevOps patří také to, že snižuje míru selhání nových verzí a zkracuje dobu mezi opravami. To je dosažené hlavně díky **automatizaci**, která zvyšuje rychlost nasazení softwaru a pomáhá k lepšímu zabezpečení. Navíc podporuje nejen rychlejší opravu chyb, ale také vylepšuje funkcionalitu. (Encyclopedia, 2022)

3 Údržba

Údržba (softwaru) je součástí SDLC, která je prováděna ihned po dodání softwaru zákazníkovi. V dnešním moderním světe se setkáváme se softwarem na denní bázi a občas také na něm závisí život ve zdravotnictví, investice na burze, řízení vesmírné sondy atd. Hlavními faktory, které jsou kladeny na dnešní software, jsou použitelnost, flexibilita, dostupnost a správnost vykonávání operací. A během životního cyklu softwaru je nutné provádět změny, aby těchto faktorů bylo dosahováno ať už samotným vylepšováním, nebo opravou chyb, které se neodstranily během vývojové fáze. (Grubb & Takang, 2007, s. 6)

Společnosti, které vyvíjí a spravují software, nyní čelí zvyšujícím se požadavkům na svůj software. Zákazníci požadují nejvyšší kvalitu za co nejmenší náklady, což u poskytovatelů vyvolává soutěž a podměty pro inovace. Aby poskytovatelé uspokojili požadavky zákazníků, musí být schopni daný software vyvinout a pak musí mít dobře nastavené vnitropodnikové procesy na implementaci nových požadavků. (April a kol., 2005, s. 197)

Na začátku je důležité představit, v jakém kontextu denní činnost údržby probíhá, tedy s jakými stranami (odděleními) údržba jedná na denní bázi.

1. **Zákazník a uživatelé** (anglicky *Customers and Users*)
   Prvním místem, kde údržba začíná, jsou zákazníci a uživatelé, s nimiž manažer domlouvá fungování služby. Diskutuje zejména o rozpětí poskytovaných funkcí, ceně, spokojenosti zákazníka a o podmínkách obsažených v SLA. Po sepsání SLA může zákazník plně využívat služeb údržby na denní bázi.

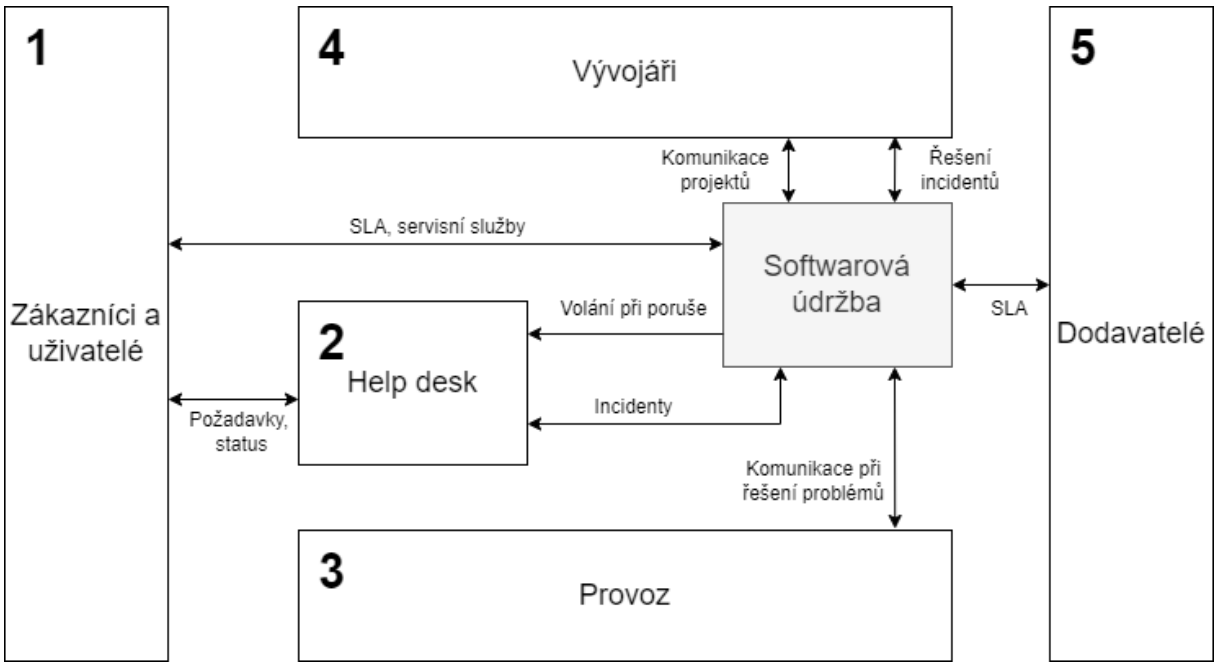
2. **Help desk**
   Dalším bodem je *Help desk*, což je centralizovaný útvar v rámci společnosti, který pomáhá řešit problémy nebo dotazy ze strany zákazníků. Zde je zajištěna komunikace se zákazníkem, prvotní řešení problémů a případně vložení tohoto problému do integrovaného systému poskytovatele a delegování na kompetentní oddělení.

3. **Provoz** (anglicky *Computer Operations*)
   Provozní oddělení v rámci údržby řeší problémy spojené s hardwarem, sítí a platformou. Provádí operace, jako jsou například zálohy systémů, obnovení do určitého bodu provozu nebo administrativní stránku systému. V rámci DN jde např. o různé mechanické problémy s bankomaty, komunikace systémů nebo nasazení různých verzí softwaru.

4. **Vývojáři** (anglicky *Software Development*)
   Vývojáři se starají o softwarovou stránku. Implementují konkrétní části kódu. Vývojáři musí mít dobré analytické schopnosti a být schopni rychle řešit zadané problémy. Nedílnou součástí jsou také dobré komunikační a organizační schopnosti, aby mohli úspěšně spolupracovat s ostatními členy týmu.

5. **Dodavatelé** (anglicky *Suppliers*)
   Posledním místem, se kterým se manažer údržby setkává, jsou dodavatelé, kteří dodávají hardware nebo software třetích stran. Pokud se nejedná o problém ze strany poskytovatele, ale vše poukazuje na problém způsobený na straně dodavatele, je nutné mu předat všechny informace a delegovat problém na oddělení podpory na základě domluvených SLA. Všechny tyto vztahy popisuje obrázek 8. (April a kol., 2005, s. 200-201)

Obrázek 8: Kontinuum softwarové údržby



Zdroj: (April a kol., 2005, s. 199), zpracováno autorem

Projektový tým údržby se skládá ze čtyř základních profesí. Patří mezi ně manažeři, analytici, designeři a programátoři. Hlavní úlohou **manažerů** je dělat správná rozhodnutí a řídit proces údržby. Provádějí také odhady času a úsilí potřebného na jednotlivé implementace. Musí mít znalosti o fungování systému, aby mohli provádět adekvátní a správná rozhodnutí, avšak nemusí znát podrobnou implementaci a kód modulů. K implementaci a technické znalosti modulů slouží profese **programátorů**, kteří provádějí změny v kódu. Mají znalosti o závislosti modulů na ostatních systémech včetně jejich podrobné funkcionality a implementace. Dalšími jsou **analytici**, kteří jsou zodpovědní za návrhy funkcionalit softwaru a porozumění řešeného problému. U této profese se předpokládá znalost externího (normy, vyhlášky, zákony, podnikání zákazníka atd.) i interního (vnitřní prostředí společnosti, procesy atd.) prostředí. Poslední profesí jsou **designeři**, kteří navrhují design systému a jeho detaily. Systémový design je návrh komponent, funkčností, datových struktur a vztahů mezi těmito komponentami, kde detailní design je zaměřen již na určitou komponentu. Jsou zde popsány také jednotlivé algoritmy, reprezentace

dat atd. Prací designera je zajistit a navrhnout fungování systému jako celku. (Niessink & van Vliet, 2000, s. 103-106)

3.1 Fáze softwarové údržby

Fáze údržby jsou velmi podobné jednotlivým fázím SDLC. Tyto fáze napomáhají ucelenému postupu v jednotlivých inkrementech údržby, mohou být samozřejmě upraveny podle podmínek dané společnosti nebo týmu. Mezi jednotlivé fáze patří:

Identifikace a trekování

Týkají se činností, které identifikují nový změnový požadavek na systém nebo požadavek na údržbu. Jak dále bude popsáno v následující kapitole ITIL, požadavek může generovat uživatel, či může být generován automatickým systémem. V této fázi se identifikuje, o jaký typ údržby se jedná, a zjišťují se další základní údaje, pro zpracování.

Analýza

Zde se kontroluje dopad na fungování systému včetně možného bezpečnostního rizika. Pokud je dopad velice závažný a ohrožuje běžný provoz zákazníka, hledá se alternativní řešení. Po analýze je k dispozici soubor specifikací neboli požadavků na změnu. K této fázi také patří odhad nákladů na požadované změny.

Design

Na základě výstupu z předchozí fáze se navrhují nové modely/změny včetně jejich testovacích scénářů pro validaci za účelem ověření funkčnosti nově implementovaných částí.

Implementace

Nové moduly jsou vyvíjeny na základě návrhu. Uplatňuje se zde například agilní metoda pro přírůstek kódu. Během této fáze se také provádějí jednotkové testy, které jsou určeny pro testování dílčích částí kódů. Provádí je každý programátor sám, nebo oddělení QA.

Systémové testování

Když jsou jednotlivé části naprogramovány, je nutné ještě ověřit, zda nejsou problémy mezi stávajícími moduly nebo mezi subsystémy, v nichž se neprováděly změny, a software jako celek vykazuje normální součinnost. Tyto testy se nazývají integrační.

Akceptační testování

Po integračních testech přicházejí na řadu testy akceptační, kdy je software předložen zákazníkovi a ten může vznést připomínky nebo nové požadavky, které budou zaznamenány k řešení v další iteraci údržby.

Nasazení

Po přijetí zákazníkem je mu software nasazen pomocí aktualizačních balíčků nebo reinstalací celého systému. Pokud se jedná o větší, nové, nebo složitější změny, probíhá

zde ještě případné školení nebo tvorba uživatelské dokumentace, jak s novým softwarem pracovat. (Tutorials Point, 2022)

3.2 Kategorie údržby

Kromě jednotlivých fází údržby existují také čtyři základní kategorie údržby. Každá z kategorií je důležitá pro správné fungování procesů údržby a žádná z nich nemůže být zanedbávána nebo opomíjena, jinak by to mělo kritické následky. Zdroj (Niessink & van Vliet, 2000) rozděluje softwarovou podporu na následující kategorie:

Opravná softwarová údržba (anglicky *Corrective Software Maintenance*)

Opravná softwarová údržba je to, co se vybaví, když se řekne slovo údržba, zabývá se totiž opravou chyb a závad, které by mohly poškodit poskytovaný software. Spadá sem design, logika a samotný kód. Podnět pro opravu chyb přichází většinou nahlášením nějaké závady zákazníkem, avšak opravná údržba se tomu snaží zabránit již nalezením chyby dříve, než doputuje k zákazníkovi či uživateli. (Cast Software, 2022)

Pro vývojáře je tento druh údržby stále časově nejnáročnější a prováděný neustále. Pokud je řešeno příliš mnoho chyb, měli by se vývojáři zaměřit na vylepšení svých dovedností, kvality kódu, lepšího testování nebo odstranění technických překážek. (Omeyer, 2021)

Preventivní softwarová údržba (anglicky *Preventative Software Maintenance*)

Preventivní softwarová údržba se zaměřuje na budoucnost, funkčnost softwaru musí sloužit tak dlouho, jak to jen půjde. Může se jednat o malé změny v současnosti, které ale mohou mít velký dopad v budoucnosti. Účelem je hledat tzv. „latentní chyby“, chyby zatím zanedbatelné, které ale mohou přerůst v „efektivní chyby“ v budoucnu, kde mohou mít několikanásobný dopad na fungování softwaru. (Thales Group, 2023)

Tato údržba napomáhá snižovat riziko a napomáhá softwaru stát se více stabilním, pochopitelným a udržovatelným. Zahrnuje změnu dokumentace, optimalizaci již napsaného kódu a restrukturalizaci kódu. (Cast Software, 2022)

Zdokonalovací softwarová údržba (anglicky *Perfective Software Maintenance*)

Zdokonalovací softwarová údržba cílí na zlepšování již obsažených vlastností a požadavků na software za účelem vylepšení hodnoty. Za cíl si dává vývoj nových komponent a vlastností softwaru, které obohatí zkušenost zákazníka či uživatele. Je proto důležité od zákazníků, např. díky dotazníkovému šetření, zjistit, které nové vlastnosti by si přál zakomponovat do vyvíjeného softwaru. Mohou se zde nabídnou i takové vlastnosti, které přímo nesouvisí s hlavní činností softwaru, ale bude se jednat o služby dodatkové. Naopak nepoužívané metody je nutné odstranit nebo je označit za „zastaralé“ a v nějaké budoucí verzi odstranit. (Cast Software, 2022)

Mezi takové změny může patřit např. vylepšení rychlosti a výkonu, zlepšení UI (= *User Interface*), přidání nových funkcionalit atd. Podle zdroje (Merrill, 2022) tvoří až 50 % všech aktivit softwarové údržby. Jedná se o **evoluční** proces softwaru.

Adaptivní softwarová údržba (anglicky *Adaptive Software Maintenance*)

Adaptivní softwarová údržba přichází na řadu, když se prostředí softwaru mění. Změnou prostředí se myslí např. změna hardwaru, operačního systému, úložiště, závislých komponent, licence, legislativy atd. Přizpůsobení těmto změnám začíná vydáním update verze softwaru zákazníkovi, aby podporovala tyto změny ve struktuře. Tím se mohou změnit některé procesy, jako je např. platba nebo zpracování dat. Je nutné mít také na mysli, že pokud bude poskytovatel provádět změny prostředí, ovlivní to i další komponenty nabízeného softwaru. (Cast Software, 2022)

Je velmi důležité předpovídat změny, které musí být provedeny. Je také dobré provést změny co nejrychleji, tedy ještě předtím, než na daný problém narazí zákazník. (Merrill, 2022)

Po přečtení si možná říkáte, že adaptivní a preventivní softwarová údržba jsou si dost podobné, proto autor představí hlavní rozdíly těchto dvou přístupů. U preventivní softwarové údržby je přirozená **evoluce softwaru**, kdy se poskytovatel snaží mít software funkční a také udělat co nejlepší dojem na zákazníka. Adaptivní softwarová údržba se snaží dosáhnout uspokojení požadavků na vyvíjený software. Je přirozené, že software roste a na trh jsou uváděny nové technologie, proto se musí daný software **adaptovat**, aby byl konkurenceschopný. (Merrill, 2022)

Všechny tyto typy údržby udržují software použitelný a přinášejí hodnotu, proto má softwarová údržba aspekty **služby** — přináší je aktivity, jako napravování chybového chování nebo implementace nové funkcionality. (Niessink & van Vliet, 2000, s. 106)

Jedna z největších výzev pro softwarové inženýry je v managementu řízení těchto změn, kdy je nejvíce vynaloženého času a úsilí zaměřeného na softwarovou podporu. Z průzkumů vyplývá, že 40 až 70 % finančních nákladů na celý životní cyklus softwaru je pouze z jeho údržby. (Grubb & Takang, 2007, s. 6)

3.3 Náklady na údržbu

Náklady na údržbu vycházejí z takzvaných celkových nákladů na vlastnictví (anglicky *Total Cost of Ownership*), které zahrnují celkové náklady během celého cyklu SDLC. Patří sem i nepřímé náklady, např. zaškolování nebo přijímání klientů. Po nákladech na implementaci softwaru, tj. do vydání první verze, přicházejí právě náklady na údržbu, které u většiny projektů dosahují 70 až 90 % z celkových nákladů na vlastnictví softwaru, a např. studie Dehaghani &

Hajrahimi, (2013) ukazuje, že tyto náklady dále rostou ve prospěch právě nákladů na implementaci. (Andreev, 2022) Faktory, které ovlivňují celkové náklady na údržbu, se primárně rozdělují na dvě kategorie.

3.3.1 Technické faktory

První kategorií jsou technické faktory, které souvisejí s technologií, hardwarem nebo softwarem daného produktu.

- **Programovací jazyky**
  Většina podniků vyvíjí software ve vyšších programovacích jazycích, které mají velkou míru abstrakce, jež umožňuje více přemýšlet o kódu jako člověk než jako počítač. To umožňuje rychlý, jednodušší a bezchybový vývoj, který šetří náklady a čas. Zdrojový kód je také menší a více přenositelný (využitelný) pro další procesy. Mezi takové programovací jazyky patří např. Java, Python, C++, PHP apod. Nevýhodou těchto vyšších jazyků je, že se musí kompilovat do nižších jazyků.

- **Softwarová validace & testování**
  Po vývoji nějaké nové komponenty, nebo celého programu je nutné software zvalidovat a otestovat, dříve než se nasadí nebo předá zákazníkovi. Pokud se tyto kroky neudělají, je vyšší pravděpodobnost budoucích problémů a tím i výše nákladů na jejich opravu. Proto by měly být součástí procesu vývoje.

- **Dokumentace a normy**
  K vývoji softwaru patří i tvorba dokumentace. Pokud se píše dokumentace od začátku do konce, může snížit náklady na údržbu, jelikož zákazníci nebo další vývojáři budou vědět, jak jednotlivé funkce softwaru fungují, a budou je moci správně ovládat nebo použít.

- **Správa konfigurace**
  Vedení dokumentace a zajišťování konfigurace může také přispět ke změně nákladů, jelikož efektivní správa konfigurace může náklady na údržbu snížit nebo v případě nesprávného postupu naopak zvýšit. Konfigurace může zahrnovat například nastavení operačního systému, sítě, databáze, webového serveru a dalších softwarových prvků.

- **Vývojářská platforma**
  Platforma, na které je software vyvíjen, je také jeden z faktorů, které mohou zvyšovat náklady na údržbu. Nejlevnější jsou platformy, které jsou zdarma k dostání a zároveň jsou velice oblíbené, jelikož není problém najít vývojáře, který s touto platformou pracuje. Dále také záleží na tom, jaký typ aplikace je vyvíjen, většinou platí, že údržba webové aplikace je levnější než např. u mobilní.

- **Hardwarová stabilita**
  Kromě softwaru je důležitým faktorem ovlivňujícím cenu také stabilita hardwaru. Použití spolehlivého hardwaru může snížit náklady na údržbu a prodloužit životnost zařízení, a to vede ke snížení celkových nákladů. Pokud je hardware nestabilní a nevyhovuje požadavkům softwaru, může to vést k častým výpadkům a chybám, což vede k prodlevám a ztrátám zdrojů. Důkladná údržba hardwaru a jeho pravidelné aktualizace jsou tedy nezbytné pro dosažení úspěšné softwarové údržby.

- **Velikost databáze**
  Jedná se o faktor, kde se hlavně sleduje, jak propojenou a komplexní databázi k aplikaci provozovatel poskytuje. Při změně požadavků zde záleží, jak velký zásah se bude muset provést. Komplexnější databáze potřebují větší náklady, jelikož je zde více tabulek a dat, které je třeba upravit, což vyžaduje více času a úsilí (práce).

- **Modulovanost**
  Rozdělení softwaru do modulů umožní měnit kód jednoho modulu nezávisle na druhém a tím snížit obtížnost a optimalizaci implementace. (Kushnir, 2021; Nagar, 2022a)

3.3.2 Netechnické faktory

Druhou kategorií jsou netechnické faktory, které spočívají v okolních vztazích, pracovních poměrech, postupech atd.

- **Stabilní tým**
  Základem údržby je tým, který vykonává všechny její kroky. Údržba softwaru se stává nezvladatelnou, pokud je tým vývojářů malý. Tým, který je stabilní, může lépe plánovat a spravovat údržbu softwaru, což vede k menšímu výskytu chyb a delším intervalům mezi opravami. To zase zvyšuje spokojenost uživatelů a snižuje náklady na údržbu softwaru.

- **Softwarový rozsah**
  Pokud je rozsah softwaru dobře definován a pochopen, umožňuje to týmu přesnější plánování a správu údržby. Dobře definovaný softwarový rozsah zvyšuje pravděpodobnost úspěšného dokončení údržby včas a s vynaložením nejnižšího možného úsilí. Naopak nejasný nebo nekonzistentní rozsah může vést k častým změnám v požadavcích a způsobovat prodlevy, což může vést k neuspokojivým výsledkům a ztrátám zdrojů.

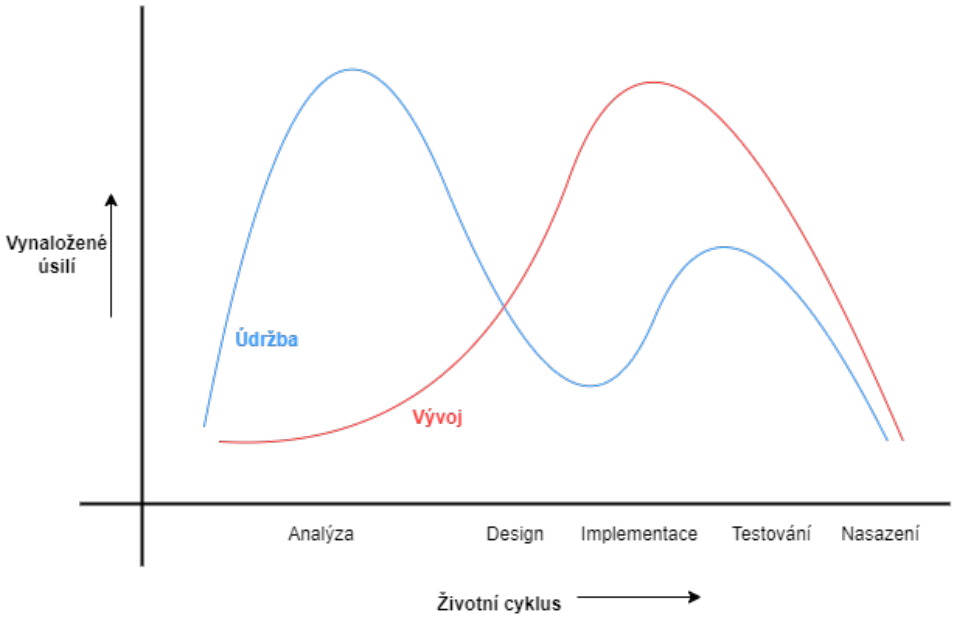
- **Fáze SDLC**
  Náklady také velice závisí na tom, v jaké fázi SDLC se software nachází. V raných fázích, jako je plánování a analýza, jsou náklady na údržbu obvykle poměrně nízké. Nicméně náklady se zvyšují v průběhu fáze návrhu a implementace, kdy jsou prováděny hloubkové změny v kódu. Ve fázi testování jsou náklady na údržbu spojené s opravami chyb

a odstraňováním nedostatků. Po vydání softwarového produktu jsou náklady na údržbu nejvyšší, protože mohou vznikat problémy až v průběhu provozu a je třeba je řešit.

- **Čas a úsilí (anglicky *Time & Effort*)**
  Jedná se o nejvíce důležitou položku nákladů. Tento faktor je závislý na složitosti projektu, náklady rostou zejména, pokud je projekt komplexnější. Projekt může být unikátní a požadavky na něj budou pro tým vývojářů nové, takže budou potřebovat více času i úsilí na jejich implementaci. Samozřejmě čím více času se budou údržbou zabývat, tím více porostou mzdové náklady.

  Obrázek 9 poukazuje na rozdíl úsilí mezi fází údržby a vývojem softwaru v rámci SDLC.

Obrázek 9: Čas a úsilí



Zdroj: (Grubb & Takang, 2007, s. 75), zpracováno autorem

- **Místo**
  Posledním netechnickým faktorem je také místo, kde je údržba provozována. V různých částech světa jsou jiné sazby a mzdy a to také ovlivňuje, jak hodně náklady na údržbu porostou. V neposlední řadě se také může měnit přístup zákazníků a jejich technické znalosti o softwaru. (Nagar, 2022a; Nagar, 2022b)

3.4 Snížení nákladů

Cílem poskytovatele softwaru a každé jiné společnosti je maximalizace zisku ať již zvýšením tržeb, nebo snížením nákladů. V této podkapitole se autor bude zabývat právě druhým zmiňovaným přístupem, a to snížením nákladů na údržbu. Budou zde představeny základní postupy, jak je možné náklady na údržbu snížit.

Prvním postupem je zaměřit se již od začátku na budování správných procesů, založených na kvalitě. Takto nastavené, transparentní procesy pomáhají se snadným řízením, kontrolou, analýzou a odhadem činností v rámci těchto procesů a přinášejí efektivnější, skladovatelnější a úspěšnější výstup. (Andreev, 2022)

K navržení procesů slouží rámce, jako je například ITIL, COBIT nebo norma ISO20000. Prvním zmiňovaným rámcem se bude autor zabývat v následující kapitole ITIL, ve které představí jednotlivé praktiky, jež souvisejí právě s údržbou softwaru.

Další technikou jak snížit náklady na údržbu, je monitoring, kterým se snaží nabízený software/služba monitorovat nepřetržitě svůj stav v intervalech, z důvodu co nejrychlejšího zachycení problému. Při výpadku musíme hlavně brát v potaz tzv. náklady na hodinu prostoje, které se zvyšují, když software/služba nepracuje, jak má, a narušuje podnikání zákazníka. Většinou se jedná o monitorovací software běžící jako webová služba, který se snaží provolávat (anglicky *Pingovat*) komponenty poskytovaného softwaru a poskytovat pak informace o jejich stavu. Při zjištění poruchy ihned informuje o změně stavu komponenty. (Takyar, 2022)

4 ITIL

S rámcem ITIL (= *Information Technology Infrastructure Library*) se autor zabýval již ve své bakalářské práci na téma „*Prioritizace zákazníků při poskytování SW podpory*“, viz (Vyleta, 2020). Avšak tam popisoval ITIL verze 3, obsahem této práce je popsat vybrané metody v nejnovější verzi 4, která byla vydána v roce 2019 a podle které společnost DN řídí jednotlivé IT procesy.

4.1 Představení rámce

Služby jsou hlavním zdrojem hodnoty, kterou může organizace zákazníkovi poskytovat. Skoro všechny služby jsou již propojené s IT, proto je v zájmu organizace vytvořit takové podmínky, aby mohla vylepšovat, rozšiřovat a spravovat ITSM (= *IT Service Management*). Jedná se o metodiku řízení IT služeb, která se soustředí na poskytování a správu těchto služeb v organizacích tak, aby byly v souladu se strategickými cíli organizace a aby byly efektivní a spolehlivé. (Holek, 2007)

Díky rozmachu nových technologií, jako je například Cloud, integrace jako služba (IaaS), Blockchain atd., je cílem organizací implementovat tyto technologie do IT procesů, aby si zajistily konkurenční výhodu na trhu, a k tomu je nutná transformace. Kvůli tomuto tlaku na IT se přizpůsobil i ITIL v4, který v sobě zahrnuje metody Lean, Agile a DevOps. Díky těmto metodám je více zaměřený na digitální transformaci ve snaze dodat co nejlepší hodnoty zákazníkovi. (Axelos, 2019, s. 13)

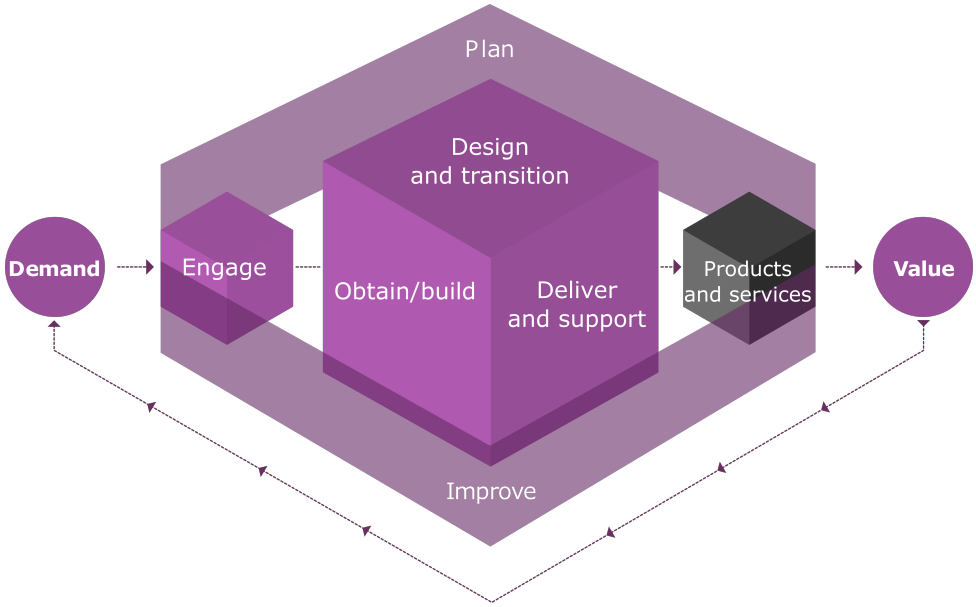
V rámci nejnovější verze ITILu je implementován také SVS (= *Service Value System*), který reprezentuje komponenty a aktivity, jež pomáhají vytvářet přidanou hodnotu zákazníkovi v IT službách. Součástí je pak i *ITIL Service Value Chain*, poskytující model pro vytváření, dodávání a neustálé vylepšování poskytovaných služeb. Pomáhá organizaci lépe reagovat na změny požadavků od svých stakeholderů. Jedná se o hodnotový řetězec, který je vyobrazen na obrázku 10.

4.2 Historie ITIL

Historii rámce ITIL autor již popisoval v bakalářské práci, proto zde bude zmíněno jen stručné shrnutí.

Rámec ITIL vznikl na konci 80. let 20. století ve Velké Británii pod vládní organizací CCTA (= *Central Computing and Telecommunications Agency*). Hlavním důvodem vzniku bylo to, že vládní IT služby nedosahovaly potřebných kvalit, a bylo nutné najít řešení, které by tuto kvalitu vylepšilo a navíc přineslo snížení nákladů na jejich dodání. Hlavní myšlenkou rámce je, že IT služby by měly být zaměřeny na potřeby zákazníka, a to by mělo být dosaženo kombinací

Obrázek 10: ITIL Service Value Chain



Zdroj: (Tayllorcox, 2022)

efektivních procesů a jasně definovaných zodpovědností, které vycházejí z konceptů a postupů prověřených praxí. Do roku 2014 vycházely publikace pod *Cabinet Office* (taktéž vládní instituce) a poté rámec přešel pod křídla společnosti *AXELOS*, která publikuje ITIL do současnosti. Od svého vzniku se ITIL stal standardem. Jeho terminologie byla široce chápaná a dala založit několika dalším modelům a rámcům, jako např. standardu *ISO 20000: Information Technology - Service Management*, *HP ITSM Reference Model*, *IT Process Model (IBM)* nebo *Microsoft Operations Framework* a další. Jak již bylo zmíněno, nejaktuálnější verzí je verze v4, která vyšla v roce 2019 a zaměřuje se na výstupní kvalitu a agilní metody. (Irwin, 2019)

4.3 Slovník ITIL

Jelikož je metodika plná odborných termínů, uvede autor na úvod slovník pojmů a definicí podle Hudec a kol. (2012), aby následující text obsažený v této kapitole byl srozumitelný.

- **Incident** je neplánované přerušení služby, nebo snížení její kvality.
- **Event** je jakákoliv změna stavu v rámci řízení služby.
- **Problém** (anglicky *Problem*) je příčina jednoho, nebo více incidentů.
- **Service desk** je komunikační bod mezi poskytovatelem služeb a zákazníkem.
- **Náhradní řešení** (anglicky *Workaround*) je dočasné řešení, které snižuje, nebo eliminuje dopad incidentu nebo problému.

- **Známá chyba** (anglicky *Known error*) je problém, u kterého je popsána příčina i jeho řešení.
- **Hot-fix** je rychlé řešení incidentu.

4.4 Vybrané praktiky správy služeb

V této podkapitole budou podrobněji popsány vybrané praktiky rámce ITIL, které souvisejí s tématem této diplomové práce, tedy vývojem softwaru a jeho údržby. Praktiky v rámci ITILu jsou de facto procesy, které popisují praxí ověřené postupy pro jednotlivé oblasti. Zmíněné praktiky jsou součástí řízení služeb.

4.4.1 Správa incidentů (*Incident Management*)

„Účel správy incidentů je minimalizovat negativní dopad incidentů pomocí obnovení normálního stavu služeb tak rychle, jak je to jen možné.“[3] (Axelos, 2019, s. 163).

Správa incidentů má velký dopad na zákazníka, protože zajištění hladkého a bezchybového chodu služeb je pro něj zásadní, pokud se služby vyřadí z provozu, ovlivní to zákazníkovo očekávání. Každý incident musí být zaznamenán poskytovatelem a se zákazníkem musí být ustanovená SLA. Incidenty by se měly dělit alespoň do tří úrovní podle dopadu na chod podnikání zákazníka.

Na straně poskytovatele služby je zapotřebí zajistit nutné zdroje, které se incidenty budou zabývat, a zajistit jejich efektivní čerpání. Je potřeba zajistit aby incidenty s malým dopadem nealokovaly příliš mnoho zdrojů, které by incidentům s velkým dopadem chyběly, a aby se tyto incidenty vyřešily co nejdříve.

Všechny záznamy o incidentech by měly být uloženy v centrálním systému (v bázi znalostí), aby poskytoval bázi všech nových znalostí pro nově identifikované incidenty. Pomocí toho lze určit, zda se s novým incidentem poskytovatel v minulosti již někdy setkal a nebo jestli může pomocí podobných incidentů z minulosti řešit ty současné.

Je také důležité, aby zaměstnanci, kteří zadávají incidenty do systému popsali daný problém co nejlépe, kromě všech jeho mandatorních atributů a uvedli na sebe také kontakt pro případnou pozdější komunikaci, pokud by se informace musely doplnit či aktualizovat. Incident může být vyřešen na různých stupních řízení incidentů. Záleží vždy na týmu lidí, kteří na něm pracují, a složitosti daného incidentu.

Některé incidenty si může zákazník vyřešit sám pomocí svého interního týmu, v němž má k dispozici nějaký systém báze znalostí a kompetentní zaměstnance, kteří jsou řádně vyškoleni.

[3] Vlastní překlad autora diplomové práce.

Pokud tuto úroveň zákazník nemá, je dalším stupněm řešení incidentu na straně service desku poskytovatele.

Pokud se jedná o složitější incidenty, přichází na řadu delegování na tým podpory, který nabídne řešení v podobě *hot-fixu*, stálého řešení incidentu, nebo se musí incident delegovat na další úroveň podpory. Incident se může dostat až na úroveň dodavatele služby nebo produktu, který podporuje vlastní správu incidentů.

Více komplexní incidenty dokonce potřebují založení vlastního týmu, který se bude daným incidentem zabývat a nabídne řešení. Může se skládat z řad stakeholderů, zmíněných v předchozích úrovních. Vyžaduje úzkou spolupráci za účelem poskytnutí znalostí a informací, ale hlavně efektivního vyřešení incidentu.

Pokud se incident deleguje na poskytovatele služby nebo produktu třetí strany, je nutné mít sepsanou SLA. V tomto případě je společnost v roli zákazníka a správa incidentů daného poskytovatele řeší incidenty v rámci svých procesů. Na průběh incidentů by měl být zaveden jednotný proces, podle kterého se bude za účelem zvýšení efektivity řešení a přidělení zdrojů postupovat. Může se jednat o automatizovaný sběr dat. Pokud se jedná o složitější incidenty, je spíše požadován expertní názor na věc. (Axelos, 2019, s. 163-165)

4.4.2 Monitorování a správa událostí (*Monitoring and Event Management*)

„Účelem monitorování a správy událostí je systematicky sledovat služby a komponenty služeb, zaznamenávat a hlásit vybrané změny stavu identifikované jako události. Tato praxe identifikuje a prioritizuje infrastrukturu, služby, obchodní procesy a události týkající se informační bezpečnosti a stanovuje vhodnou odezvu na tyto události, včetně reakce na podmínky, které by mohly vést k potenciálním poruchám nebo incidentům.“[4] (Axelos, 2019, s. 171).

Monitorování má za úkol spravovat eventy za účelem odhadu, minimalizování nebo eliminace negativního dopadu na podnikání zákazníka. Zaměřuje se na systematické sledování služeb a zjišťování jejich stavu, aby zjistilo, zda došlo k eventu, nebo je provoz služby v pořádku, popřípadě informovalo oddělení podpory prostřednictvím nějakého předem vybraného informačního kanálu. Monitorování by mělo být plně automatické a mělo by být prováděno aktivně i pasivně. Správa událostí se zaměřuje na zaznamenávání a správu změn od normálního stavu (eventů). Monitorování tento event zaznamená a správa událostí pak identifikuje, o jaký typ se jedná, zjistí jeho závažnost a spustí nápravná opatření.

Eventy mají různou významnost a dělíme je na informativní, varovné, nebo na výjimkové. Informativní eventy slouží pouze jako stav služby v nějakém čase a nevyžadují tak žádné opravné kroky. Varovné eventy slouží pro upozornění na negativní dopad ještě před jeho vznikem a

[4] Vlastní překlad autora diplomové práce.

umožňuje konat preventivní akce. Poslední, výjimkové eventy upozorní na negativní dopad, až když nastane. Může se jednat například i porušení normy, jako je SLA.

Tato služba je vysoce interaktivní s dalšími službami a podle jednotlivých eventů lze identifikovat, kam bude dále delegována, zda na správu problémů, nebo incidentů. (Axelos, 2019, s. 171-172)

4.4.3 Správa problémů (*Problem Management*)

„Účelem správy problémů je snížit pravděpodobnost výskytu a dopadu incidentů tím, že identifikuje skutečné a potenciální příčiny incidentů a spravuje dočasné řešení a známé chyby.“[5] (Axelos, 2019, s. 175).

Každá služba má chyby, nedostatky nebo slabiny, které mohou způsobovat tvorbu incidentů. Mnoho chyb je již identifikováno a opraveno před vydáním služby, ale nějaké chyby zůstanou neodhaleny a vytvářejí zde riziko. Tyto chyby jsou podle metodiky ITIL nazývány pojmem problémy.

Problémy jsou spjaty s incidenty, ale mezi největší rozdíly patří:

- Incidenty mají dopad na zákazníka a jeho podnikání a musí být vyřešeny, aby podnikové aktivity mohly pokračovat.
- Problémy jsou příčiny těchto incidentů. Požadují identifikování příčin, vymyšlení dočasných a nacházení dlouhodobých řešení.

Služba zahrnuje tři fáze: **Identifikace problémů** má za cíl, jak název napovídá, identifikovat a zaznamenat problémy. Dalšími aktivitami v rámci této fáze je analýza vývoje problému, odhalování duplicitních problémů kvůli zamezení vzniku incidentů, které již byly jednou řešeny, a analýza informací od dodavatelů ať už externích, nebo interních.

Řízení problémů se již zabývá samotnou podstatou problémů, analyzuje je a dokumentuje známé chyby a poskytnutá náhradní řešení. S problémy se zachází stejně jako s riziky, je k nim přiřazen dopad a pravděpodobnost vzniku, dále jsou podle těchto atributů prioritizovány a v tomto pořadí analyzovány. V rámci analýzy by se mělo postupovat stromově a hledat související příčiny, aby se přišlo na jádro problému. Pokud se náhradní řešení problému prokáže jako dlouhodobé, je zaznamenáno do databáze řešení.

Řízení chyb se zabývá správou známých chyb, které ještě nebyly vyřešeny. Pokud se najde vhodné řešení, dá podmět k tomu, aby se řešení implementovalo. Poslední činností řízení chyb je také měření zavedených náhradních řešení a kontrola, zda opravdu došlo k opravě chyby. (Axelos, 2019, s. 175-177)

[5] Vlastní překlad autora diplomové práce.

4.4.4 Řízení úrovně služeb (*Service Level Management*)

„Účelem řízení úrovně služeb je nastavit jasná cílová hodnocení služeb na základě podnikových požadavků a zajistit, že dodávka služeb je adekvátně hodnocena, monitorována a řízena s ohledem na tyto cíle.“[6] (Axelos, 2019, s. 202).

SLM (= *Service level management*) se zabývá správou služeb od stanovení požadavků se zákazníkem až po vyhodnocování sledovaných metrik a jejich případnou nápravou na požadovaný stav. Jedná se tedy o controlling v rámci služeb.

Prvním krokem je stanovení SLA, kde každá metrika by měla souviset s poskytovanou službou, aby byly účelné a nevytvářely se zbytečně. Při jejich sepisování by měly být přítomné všechny strany, kterých se to týká, a měly by být sepsány co nejjednodušeji, aby byly srozumitelné. (Axelos, 2019, s. 202-203)

Mezi nejběžnější metriky SLA patří např.:

- **Perioda podpory** (anglicky *System Support Period*) — vymezená doba, kdy se na problému bude pracovat,
- **Dostupnost** (anglicky *System Availability*) — čas, kterým se poskytovatel zavazuje, že nabízená služba bude fungovat,
- **Reakční doba** (anglicky *Responce Time*) — maximální čas potřebný pro zahájení oprav při výpadku,
- **Doba opravy** (anglicky *Fix Time*) — maximální časová lhůta pro odstranění závady,
- **Komunikace** (anglicky *Comunication*) — kanál, kterým se bude komunikovat. (Blue Partners, 2022)

SLM se zaměřuje na sledování požadavků, podmínek, připomínek a potřeb zákazníků. Cílem je pak navrhovat nové funkcionality, které by mohl zákazník využít v budoucnu nebo které potřebuje nyní. Důležité je se zákazníkem budovat vztah, aby věděl, že jeho požadavky jsou pochopeny a poskytovatel na ně bere zřetel. To SLM docílí pomocí sledování a analýzou různých zdrojů, např. zapojením zákazníků do procesu návrhu a vylepšování služby, zpětnou vazbou nebo operačními a podnikovými metrikami. (Axelos, 2019, s. 204)

[6] Vlastní překlad autora diplomové práce.

5 Shrnutí teoretické části

Zdroje zpracovávané v této práci jsou převážně v anglickém jazyce a ze zahraničních zdrojů. Tuzemské zdroje informují v obecné rovině a nemají takovou vypovídající hodnotu a aktuálnost. Mnoho zdrojů pochází z publikovaných internetových článků, webových stránek či internetových fór, v nichž se dají dohledávat nejaktuálnější informace, a zjistit, s čím se současní poskytovatelé IT služeb nejvíce potýkají. Vložené obrázky jsou většinou také v angličtině.

Základní literaturou pro psaní teoretické části byly zejména odborné články (Niessink & van Vliet, 2000) a (April a kol., 2005) z časopisu „*Journal of Software Maintenance*“, které dobře popisují personální a procesní strukturu údržby, dále (Grubb & Takang, 2007), kteří zase popisují údržbu nejen obecně, ale i jak ovlivňuje všechny úrovně vývoje softwaru. Z tuzemské literatury je to pak (Procházka & Klymeš, 2011), kteří popisují základní pojmy a metodiky se zaměřením na agilní vývoj. V kapitole ITIL je hlavním zdrojem (Axelos, 2019) jakožto metodika pro řízení IT služeb, doplněná definicemi z (Hudec a kol., 2012). Vše doplňují již zmiňované webové zdroje.

Jak z teoretické části vyplývá, vývoj softwaru je složitý a nákladný proces, kdy zanedbání jedné fáze vede k budoucímu zvyšování nákladů. Základem je zákazník, kterému bude software poskytován jako IT služba, a u něhož je třeba dbát na jeho požadavky a spojenost. Služba by měla být od počátku koncipovaná správně za účelem co nejvyšší kvality, spolehlivosti, efektivnosti a bezpečnosti. Pro vývoj se používají různé metody, které reagují na změny, jež nastávají vždy ať už ze strany zákazníka nebo evolucí softwaru. Důležitá je také spolupráce mezi jednotlivými odděleními u poskytovatele a dobré nastavení procesů, které kombinuje přístup DevOps. Po nasazení softwaru u zákazníka vývoj nekončí, ale pouze začíná cyklus podpory, který se stará o hladký a bezproblémový chod na straně zákazníka a snaží se eliminovat potencionální problémy nebo řešit již vzniklé problémy. Je to fáze, ve které se software nachází nejdéle a která je jednou z nejvíce nákladných. Pro správné nastavení IT procesů se stará metodika ITIL, která je implementována v DN.

V praktické části se autor bude zabývat popisem podpory v rámci DN včetně popisu odhadu nákladů, které chodí na oddělení *Support & Maintenance* pomocí vybraných statistických metod.

6 Predikce nákladů ve společnosti

Tato kapitola objasní SDLC v rámci DN včetně jejich modelu podpory. Dále v kapitole bude představen současný model výpočtu predikce nákladů.

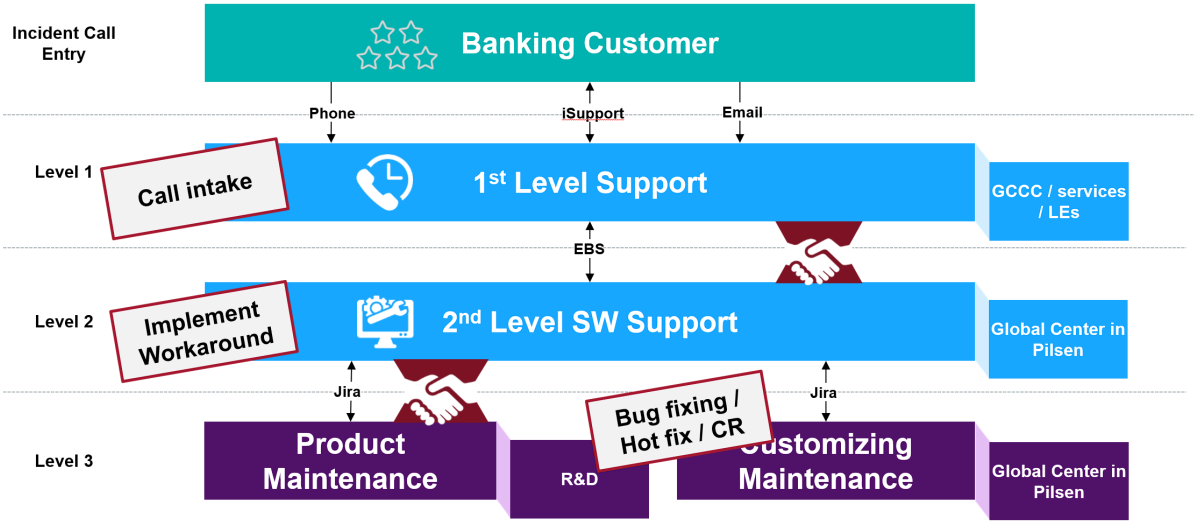
6.1 Základní činnosti podniku

Vývojový cyklus softwaru začíná u týmu *Research & Development* (R&D), který vyvíjí hlavní software (anglicky *Product*). Dále na řadu přistupuje projektový tým (anglicky *Project*), složený z vývojářů, jenž na základě požadavků a přání zákazníka dodává tzv. customizaci — produkt, ve kterém jsou např. přidány další funkce, změněn vzhled atd. podle zadání zákazníka. Posledním týmem je pak *Maintenance & Support* (M&S), který se, jak z názvu napovídá, stará o podporu a údržbu nabízeného softwaru. Je prvním styčným bodem pro zákazníka, jelikož řeší komunikaci se zákazníkem o jeho problémech. (Vyleta, 2020, s. 40)

Ve společnosti existují tři úrovně podpory (anglicky *Support Levels/Tiers*). Proces podpory začíná výskytem události (anglicky *Eventu*) na straně zákazníka. Je možné, že zákazník má na své straně zřízenou první úroveň podpory, tzv. **Tier 0**, který může událost vyřešit na základě vlastních nebo získaných znalostí, příruček atd. Pokud zákazník „nultou“ podporu nemá zřízenou nebo si s událostí sám neporadí, obrací se pomocí předem domluveného komunikačního kanálu — emailem, interním systémem, telefonicky (definovaného v SLA) na první stupeň zákaznické podpory **Tier 1**. Jedná se o tzv. *Help desk*, který poskytuje technickou pomoc zákazníkům při řešení různých problémů. V rámci společnosti se jedná o GCCC (= *Global Custemer Call Center*), které se nachází v Polsku. Zde se daný problém více zdokumentuje a případně se od zákazníka získají další, podrobnější informace a tím vznikne incident. Tímto se incident zanese do systému a stává se z něho incident (anglicky *Issue*), který musí být řešen nebo projednán. Pokud daný problém Help desk není schopen vyřešit, deleguje ho na místo s vyšší technickou podporou **Tier 2**. Tato úroveň se již nachází v Plzni. Její tým se nejdříve zabývá kontrolou a zkoumáním poskytnutých logů od zákazníka a poskytuje mu FAQ (= *First Qualified Answer*), což je první odpověď směrem k zákazníkovi k jeho problému. Zde mu sdělí, jestli je možné incident nějakým způsobem napravit, nebo se musí delegovat na vyšší úroveň a dále řešit. Pokud je náprava možná, zabývá se dále analýzou incidentu, kontroluje již zákazníkovo nastavení, komponenty atd. a snaží se najít příčinu problému. Snaží se také vymyslet, otestovat a implementovat funkčnost dočasného řešení. Pokud se i přes dílčí snahy nedokáže příčina problému eliminovat, deleguje se již na poslední a nejvyšší úroveň podpory **Tier 3**. Tato úroveň se rozděluje na dvě oddělení, PM (= *Product Maintenance*) zabývající se vývojem produktu a, CM (= *Customizing Maintenance*), zabývající se projektovými customizacemi daného zákazníka. Na této úrovni se z incidentu stává problém, který je potřeba vyřešit. Zde je řešení incidentů na velice vysoké technické úrovni a provádí se zde oprava chyb, implementace nových funkcí nebo poskytnutí dočasného řešení problému. Navržené řešení problému je zpět posláno

na nižší úroveň a otestováno oddělením QA. Pokud se řešení ukáže jako akceptovatelné, je poskytnuto zákazníkovi, který ho aplikuje pomocí opravných balíčků na svůj software, a problém je označen jako dokončený (anglicky *Resolved)*. Tento model podpory zobrazuje obrázek 11. (Vyleta, 2020, s. 40-41)

Obrázek 11: M&S model



Zdroj: (Diebold Nixdorf, 2020)

6.2 Predikce nákladů

Ve společnosti DN je nutné odhadovat náklady (anglicky *Cost*) kvůli stanovení cenové nabídky pro zákazníky. Bez odhadu nákladů a příjmů (anglicky *Revenue*) z jednotlivých projektů by společnost nebyla schopná určit cenovou politiku, která by dosahovala dlouhodobých výnosů. Pokud jde o odhad nákladů na M&S, jsou zde tři otázky, které si musí společnost zodpovědět:

1. *Co jsou hlavní spouštěče odhadů pro M&S?*
2. *Jaké informace potřebujeme, abychom mohli M&S nacenit?*
3. *Jak potřebujeme spolupracovat, aby naše nabídky za M&S byly konkurence schopné?* (Diebold Nixdorf, 2020)

Pro tyto účely existuje interní nástroj **EaFAT** (= *Effort and Financial Assessment Tool*), což je precizní výpočet založený na mnoha odhadech. Autor práce nebude popisovat podrobněji tento nástroj, uvede pouze základní myšlenku a princip. EaFAT umožňuje výpočet nákladů v jednotkách let. Základní myšlenkou je odhad fixních a variabilních nákladů. Mezi fixní náklady patří například různé licence softwaru, poradenské činnosti nebo náklady na přenos znalostí a know-how mezi týmy, které jsou v kompetenci M&S. Jedná se o náklady, které jsou většinou

pevně dané a dají se předem vypočítat s přesnou hodnotou. Hodnota variabilních nákladů se předem neví. O jejich hodnotě rozhoduje několik faktorů, které se mohou v čase měnit a jdou špatně předpovídat. Musí tedy vycházet z odhadů nebo historických dat. Základním vzorečkem pro výpočet nákladů pomocí EaFAT pro jednotlivou úroveň podpory je rovnice 1. (Krivánka, 2018)

$$v_n = \begin{cases} x_1 \times e_1 \times r_1 & \text{když } n = 1 \\ (x_n \times (1 - s_{n-1})) \times e_n \times r_n & \text{když } n > 1 \end{cases} \tag{1}$$

Zdroj: Vlastní zpracování s využitím (Krivánka, 2018)

kde: v_n ... variabilní náklady na n-té úrovni [€],
n ... úroveň podpory,
x_n ... počet ticketů na n-té úrovni [ks],
e_n (*Effort*) ... průměrný čas vynaložený na řešení jednoho ticketů na úrovni n [h],
r_n (*Rate*) ... průměrná sazba za jednu hodinu na úrovni n [€],
s_n (*Success Rate*) ... počet ticketů vyřešených na úrovni n [%].

Rovnice 2 zobrazuje výsledný vzorec pro výpočet celkových nákladů pro oddělení M&S.

$$ms = (\sum_{i=1}^{k} v_i) + F \tag{2}$$

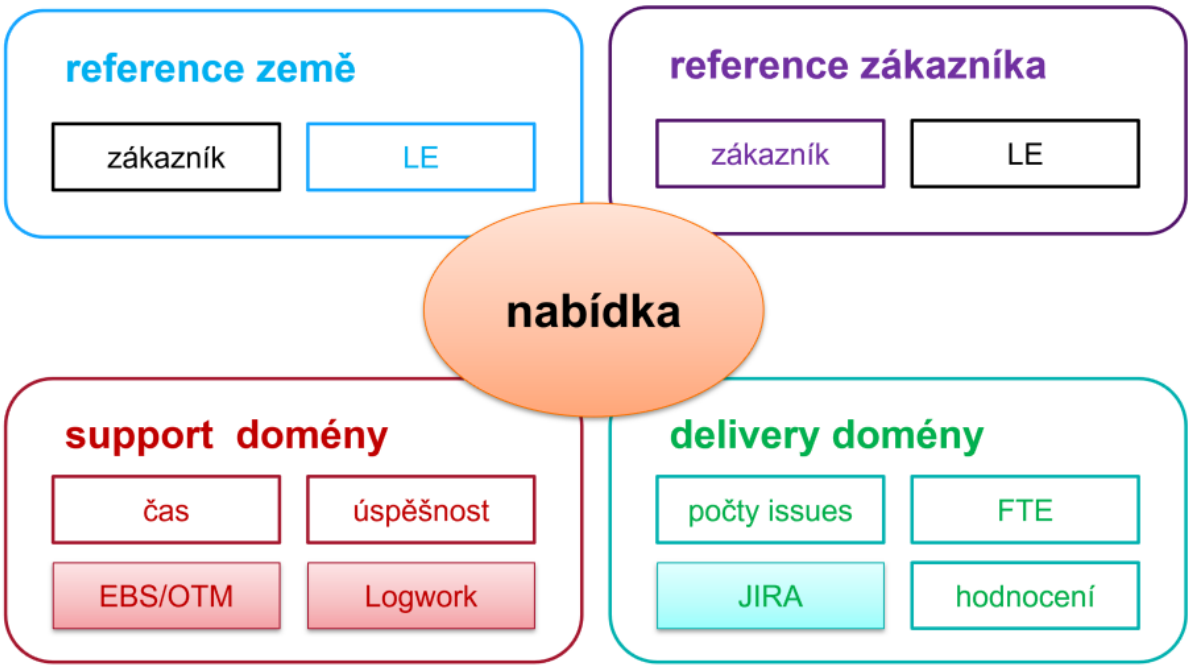
Zdroj: Vlastní zpracování s využitím (Krivánka, 2018)

kde: ms ... celkové náklady pro všechny úrovně M&S,
v_i ... variabilní náklady na i-té úrovni [€],
k ... počet úrovní podpory [ks] (většinou 3),
F(*Fixed costs*) ... fixní náklady [€].

Tato metoda je velmi citlivá na správné počáteční odvození (vstupních) proměnných x_n a e_n, které vycházejí z dalších čtyř latentních proměnných, které jsou popsány dále. Další proměnná modelu je r_n, která je daná tabulkově. V prvním kroku se vypočítají náklady (v Eurech) na první úroveň podpory. V dalších výpočtech se objevuje již i proměnná s_n, která je odvozována z dat z minulých období, a její význam je takový, že udává počet vyřešených ticketů na dané úrovni podpory. Avšak v modelu se musí vypočítat tzv. „*Failure rate* (poměr nevyřešených problémů)“, které se delegují na vyšší úroveň podpory, na nichž vznikají další náklady s novými sazbami. Po výpočtu všech úrovní podpory se přičtou fixní náklady, kterými mohou být např. platby pro konzultační týmy nebo různá školení.

Odvozované proměnné x_n a e_n vycházejí z obrázku 12. Nabídkou je myšlen odhad, kde reference země a reference zákazníka odvozují počet ticketů, tedy proměnnou x_n. Je tím myšlen specifický zákazník, se kterým má společnost zkušenosti, má dostupné informace a kde zákazník vyžaduje své služby. Support domény a delivery domény odvozují proměnnou e_n. Pod těmito skupinami se skrývá samotné dodávání služby zákazníkovi, ale také samotná funkcionalita.

Obrázek 12: Proměnné ovlivňující nabídku



Zdroj: (Krivánka, 2018)

6.3 Ovlivňující faktory

Podle interních zdrojů společnosti (Diebold Nixdorf, 2020) se na skladbě odhadovaných nákladů podílejí zejména prvky:

- **Náročnost** (složitost)
  Kolik úsilí (anglicky *Effort*) zabere implementace problému, čím větší, tím vyšší náklady na údržbu.
- **Odchylka od produktu**
  Odchylkou jsou myšleny nové funkcionality v projektu, které nejsou implementované v produktu a musí se dodělávat. Čím více je individuálních specifikací, tím déle implementace zabere, a to vede ke zvýšení nákladů.
- **Zákazník**
  Na nákladech se i podílejí zkušenosti s daných zákazníkem, tedy jak schopný tým zákazník

má. Příkladem mohou být poskytnuté informace o problému, zda je zákazník schopný si nějaký problém vyřešit sám apod. Čím méně „schopný“ zákazník, tím větší ponechané úsilí na DN a to zapříčiní vyšší náklady.

- **Kvalita projektu**
  Kvalita projektů závisí na vývojářském týmu, který projekt vyvíjí. Pokud tým během testovací fáze objeví velký počet defektů, jsou pak náklady na budoucí odstranění problémů menší. Většinou závisí na zkušenostech daného týmu, předpokládá se, že pokud testy dělají senioří vývojáři, kvalita projektu je vyšší.

- **Počet zařízení**
  Zákazníci jsou různě velcí a mají různý počet zařízení, kde software běží, čím více zařízení zákazník má, tím vyšší náklady, jelikož se musí dělat rozsáhlejší distribuce poskytovaného softwaru. Také záleží na typu zařízení, kdy pro různé typy zařízení se musí instalovat různé ovladače, a tím se také zvyšují náklady, jelikož roste vynaložené úsilí.

- **Životní cyklus**
  Velice také záleží na životní fázi softwaru (podle SDLC). Je levnější najít chyby ve vývojové nebo testovací fázi, než kdyby byl software již nasazen u zákazníka. Podle zkušeností, je krátce po nasazení u zákazníka nacházeno více problémů, než když zákazník má software nasazený dlouhodobě.

- **Doba trvání smlouvy**
  Časově delší smlouvy smluvené se zákazníkem jsou pro DN výhodnější, jelikož se náklady na údržbu rozprostřou konstantně napříč smluvenými lety, avšak první roky jsou vždy nákladnější (viz bod Životní cyklus).

- **Licence softwaru**
  Licencí je myšlen obchodní model. Zákazník může platit ve dvou variantách:

  1. Předplatné (anglicky *Subcription*) — První variantou je předplatné, kdy zákazník platí během měsíce/roku částku za poskytované služby. V téhle variantě je nabízena nejnovější verze produktu a při vydání nové verze na ni automaticky zákazník přechází. V této variantě rostou náklady kvůli časté změně verzí.

  2. Prodej licencí — druhou variantou je prodej licencí, kdy si zákazník koupí licenci na prověřenou verzi, kterou pak následně využívá. Zde se hlavní verze produktu nemění a tím pádem jsou náklady na údržbu menší.

- **SLA**
  Záleží také na smluvených podmínkách v rámci SLA. Pokud jsou domluveny kratší (rychlejší) časy na odpověď, jsou náklady vyšší, jelikož oddělení podpory musí řešení problému prioritizovat a nasadit na něj více zaměstnanců.

- **Celkové náklady**
  Jedná se o celkové náklady na nasazení produktu u zákazníka. Pokud má zákazník složitou architekturu systémů, může se nasazení časově i personálně prodloužit, tudíž to vede ke zvýšení nákladů.

7 Metodika

V této části diplomové práce autor navrhne statistické metody pro analýzu dat o celkově stráveném čase (úsilí) v rámci incidentů společnosti DN včetně stanovení výzkumných otázek. Nejdříve bude popsán problém, který tato diplomová práce řeší společně s návrhem vhodných statistických metod, relevantních pro řešený problém a zajišťujících správnost a spolehlivost výsledků.

7.1 Popis cíle a výzkumných otázek

Cílem praktické části a celé práce je pomocí statistických metod provést analýzu poskytnutých dat a provést predikci nákladů na incident pomocí odhadu doby trvání jeho řešení. Součástí je i hledání vztahů mezi proměnnými a jejich interpretace. Prací autora je pak posoudit, jestli lze na základě dat v nějaké rozumné kvalitě predikovat hodnoty úsilí daného incidentu. Samotná úspěšnost predikce bude porovnána pomocí několika metrik. V rámci praktické části bude také tvorba aplikace, která v sobě zahrnuje formulář pro zadávání nezávislých proměnných (proměnné incidentu). Na základě těchto zadaných proměnných pomocí vybraných metod predikuje úsilí incidentu. Takto zaměstnanci z DN získají hrubou představu o odhadu nově příchozího incidentu, což jim poskytne možnost lépe předpovídat výši nákladů na projekt. Byly stanoveny následující výzkumné otázky:

1. *Jakými faktory v rámci incidentu je ovlivňován celkově strávený čas?*

2. *Je možné na základě dostupných dat a zvolených metod predikovat celkově strávený čas na incident?*

3. *Lze pomocí metod predikce odhadnout celkově strávený čas na celý projekt za určitý časový interval?*

První výzkumná otázka bude hledat nezávislé proměnné, které nejvíce ovlivňují predikci závislé proměnné. Nalezení těchto hodnot může vést k lepšímu porozumění vztahům mezi proměnnými. Pro nalezení se mohou použít metody lineární regrese nebo diskriminační analýzy, podle charakteru dat. Druhá stanovená výzkumná otázka bude ověřovat kvalitu vybraných modelů na poskytnutých datech. Pro ověření kvality vybraných modelů je možné použít koeficient korelace nebo F-skóre, opět podle charakteru dat. Plánování nákladů pro určitého zákazníka se odvíjí od celkově stráveného času na řešení incidentů. K tomuto účelu byla stanovena třetí výzkumná otázka, která na základě vytvořené aplikace vyhodnotí celkově strávený čas incidentů (pro vybraného zákazníka) zvolenými metodami a porovná tento čas se skutečně vynaloženým časem. Výsledky budou vyhodnoceny pomocí t-testu.

7.2 Návrh postupu

Podle charakteru poskytnutých dat, která budou analyzována v následující kapitole, se bude závislá proměnná úsilí predikovat nebo klasifikovat. Pokud mezi daty bude dostatečný počet kardinálních proměnných, může se využít vícerozměrná lineární regresní analýza (MLRA = *Multidimensional Linear Regresion Analysis*) pro predikci hodnoty závislé proměnné. V rámci tohoto modelu se také určí, jestli vůbec existují lineární vztahy mezi závislou a nezávislými proměnnými. Pokud předpoklady pro MLRA nebudou splněny, bude se muset model upravit a použít lineární diskriminační analýza (LDA = *Linear Discriminant Analysis*) a kvadratická diskriminační analýza (QDA = Quadratic Discriminant Analysis), pro zjištění klasifikace kategorie závislé proměnné. Kardinální hodnoty závislé proměnné budou muset být rozděleny na kategorie. Pro zmíněné modely budou výsledky porovnány s metodami strojového učení, kterými jsou náhodný les a neuronová síť. Podrobnější popis a postup jednotlivých metod bude popsán v následujících kapitolách. Všechny hypotézy v této práci budou měřeny na zvolené hladině $\alpha = 0,05$, což je standardní hodnota pro statistické testy.

8 Explorační analýza

Tato kapitola se zabývá explorační analýzou dat (EDA = *Exploratory Data Analysis*), v rámci které bude popsán sběr dat z interní databáze společnosti, čištění dat za účelem odhalení chybějících hodnot, duplicitních záznamů, atypických hodnot a dalších nečistot v datech. Součástí kapitoly je také vizualizace dat pro rychlý přehled o datových strukturách a nakonec popisná statistika proměnných.

8.1 Popis sběru dat

Data pro tuto diplomovou práci byla poskytnuta společností DN, která je uchovává v ticketovacím softwaru JIRA. Data jsou uložena v databázovém systému a k jejich přístupu se využívá psaní dotazů v jazyce JQL (= *Jira Query Language*). Přístup k systému je dostupný pouze z interní sítě (musí být založen účet), takže k němu mají přístup pouze autorizovaní zaměstnanci, a proto byla data poskytnuta přímo v prostorách společnosti na jednom z počítačů. Po spuštění dotazu byla jednotlivá data z databáze vyfiltrována a exportována do souborů CSV (= *Comma Separated Value*), který je běžně používaným formátem pro úschovu a přenos dat, jednotlivé záznamy jsou vždy na jednom řádku a k rozdělení konce záznamů se nejčastěji využívá středník.

Data jednotlivých incidentů jsou z časového rozmezí deseti měsíců (od května 2022 do února 2023). Data obsahují všechny atributy incidentů a pro následující analýzu byla vyfiltrována a vybrána pouze některá, což je popsané v následující podkapitole. Celkově bylo ze systému poskytnuto 3 940 incidentů. K těmto incidentům byla z dalšího interního, blíže nespecifikovaného systému ještě získána data o celkovém čase a celkových nákladech ve formátu excelovské tabulky (přípona *.xlsx*). Data byla sloučena pomocí identifikačního klíče do jednoho souboru, který je vstupem pro následující analýzu.

8.2 Popis dat

Jak již bylo zmíněno, byly získány incidenty se všemi proměnnými, celkově jsou data popsána 446 proměnnými. Mnoho z těchto proměnných jsou buď prázdné, nebo se k analýze nehodí, proto byly po domluvě s vedoucím z DN — Ing. Davidem Kriványkem (osobní komunikace, 15. 11. 2022) vybrány proměnné vhodné pro analýzu, které budou popsány dále. Některé proměnné také musely být vytvořeny z původních dat a jsou popsány v kapitole Kódování dat pro model.

8.2.1 JIRA incidenty

V této podkapitole jsou představeny vybrané proměnné získané z obou systémů. Proměnné jsou uvedeny v tabulce 1, ve které jsou více popsané, včetně typu proměnné odvozené pomocí zdroje (Hendl, 2015, s. 48). Vysvětlivky pro jednotlivé zkratky typu proměnných:

- KS — Kardinální spojitá
- NK — Nominální kategorická
- ND — Nominální dichotomická
- NO — Nominální ordinální

Tabulka 1: Popis dat

Název proměnné (anglicky)	**Název proměnné (česky)**	**Typ proměnné**	**Jednotka (anglicky)**
Current support team	Současný tým podpory	NK	Category
Currently affected numbers of machines	Počet ovlivněných strojů	NO	Category
Customer	Zákazník	NK	Category
Frequency of occurence	Četnost výskytu závady	NO	Category
Impact	Dopad	ND	Critical/Non-critical
Is business partner	Business partner	ND	Yes/no
Priority	Priorita	NK	Category
Product	Produkt	NK	Category
Project phase	Projektová fáze	NK	Category
Region	Region	NK	Category
Reproducibility	Reprodukovatelnost	ND	Yes/no
Spent time	Strávený čas (úsilí)	KS	Minutes (min)
Status	Stav	NK	Category
Urgency	Závažnost	NO	Category

Zdroj: Vlastní zpracování, 2023

Dále budou jednotlivé proměnné popsány dopodrobna včetně popisu obměn (kategorií), pokud se jedná o nominální proměnnou:

- **Current support team** — odpovědný tým za řešení incidentu (každý tým má přidělený nějaký produkt):
  - *team-pm-banking-sw-vcps* — ProFlex4, ProCash, ProTopas, ProChip,
  - *team-pm-banking-sw-vocms* — ProView (Vynamic View),
  - *team-pm-banking-sq-vss* — Vynamic Security,
  - *team-pm-banking-sw-vctees* — Transaction Middleware, PC/E,
  - *team-pm-banking-sq-proakt* — ProAKT.
- **Currently affected numbers of machines** — počet ovlivněných strojů:
  - *1*; *2-10*; *11-100*; *101-1000*; *more than 1000*.
- **Customer** — zákazník, kterému projekt patří. Kategoriemi jsou zde jednotliví zákazníci.
- **Frequency of occurence** — četnost výskytu incedentu:

  - *once* — jednou,
  - *less than once a week* — méně než jednou za týden,
  - *once a week* — jednou za týden,
  - *daily* — denně,
  - *always* — vždy.
- **Impact** — vyjadřuje dopad incidentu na podnikání zákazníka:
  - *non-critical* — dopad není kritický,
  - *critical* — dopad je kritický.
- **Is business partner** — atribut vyjadřující, zda se jedná o business partnera:
  - *yes* — je,
  - *no* — není.
- **Priority** — značí závažnost incidentu:
  - *low* — nízká priorita,
  - *medium* — střední priorita,
  - *high* — vysoká priorita,
  - *highest* — nejvyšší priorita.
- **Product** — základní verze produktu.
- **Project phase** — fáze projektu, ze které incident pochází (jsou popsány v kapitole Fáze softwarové údržby):
  - *production* — projekt je již v produkci u zákazníka (nejvyšší dopad),
  - *pilot* — projekt je ve zkušební fázi,
  - *UAT* — projekt se testuje na straně zákazníka,
  - *certification* — projekt je ve fázi certifikace,
  - *SIT* — projekt je ve fázi systémových testů,
  - *internal QA* — projekt je ve fázi testování oddělením QA.
- **Region** — světový region, ze kterého incident pochází:

  - *APAC* — Asijský Pacifik,
  - *EMEA* — Evropa, Střední východ a Afrika,
  - *LATAM* — Latinská Amerika,
  - *NA* — Severní Amerika.
- **Reproducibility** — reprodukovatelnost incidentu:
  - *yes* — reprodukovatelný,
  - *no* — nereprodukovatelný.
- **Spent time** — celkový čas strávený řešením incidentu na straně poskytovatele.
- **Status** — stav incidentu k datu stažení dat:
  - *resolved* — řešení poskytnuté a již schválené zákazníkem,
  - *cenceled* — zamítnutý,
  - *solution provided* — řešení poskytnuté (ale ještě neschválené) zákazníkem,
  - *under GES investigation* — rozpracovaný,
  - *waiting for [strana]* — čekání na některou ze stran, než poskytne více informací. (Např. *waiting for GE support*, *Waiting for Reporter* atd.).
- **Urgency** — naléhavost incidentu ze strany zákazníka:
  - *low* — nízká naléhavost,
  - *medium* — střední naléhavost,
  - *high* — vysoká naléhavost,
  - *highest* — nejvyšší naléhavost.

8.2.2 Kódování dat pro model

V této kapitole jsou popsány proměnné, které musely být pro model nějak upraveny, nebo vypočítány. Důvodem je to, že některá data musela být anonymizována nebo transformována pro přesnější výpočet v analýzách.

Business partner

Tato proměnná byla převedena na dichotomickou nominální proměnnou „Is business partner“, jelikož nás bude zajímat jen informace, zda incident založil business partner, nebo

ne. Nominální kategorická proměnná se tedy promění v nominální dichotomickou podle následujícího pseudokódu:

```
if ([Business partner] == NULL)
    write FALSE
else
    write TRUE
```

Customer

U této proměnné proběhlo sloučení mezi „Customer from EBS“ a „Customer/s“ na základě podmínky. „Customer from EBS“ je vyplněn jen v případě, že incident reportoval partner DN a „Customer/s“ pokaždé. Pokud je tedy hodnota u „Customer from EBS“ prázdná, bere se hodnota z pole „Customer/s“, jinak se bere hodnota „Customer from EBS“. Nakonec byl tento sloupec převeden na číselné kódy, aby byla zajištěna ochrana citlivých dat. Podmínku pro vytvoření výsledné proměnné můžeme zapsat následujícím pseudokódem:

```
if ([Customer from EBS] == NULL)
    take [Customer/s]
else
    take [Customer from EBS]
```

Product

Kódování této proměnné je na stejném principu jako předchozí proměnná, protože vznikla sloučením na základě podmínky mezi sloupci „Product from EBS“ a „Product/s“. Pro ochranu citlivých dat byla také převedena na číselné kódy. Podmínku zobrazuje následující pseudokód:

```
if ([Product from EBS] == NULL)
    take [Product/s]
else
    take [Product from EBS]
```

Možná jste si všimli, že mezi proměnnými se nevyskytuje žádná nákladová proměnná, je to z toho důvodu, že variabilní náklady jsou odvozené od proměnné „Spent time“, která se násobí peněžní sazbou určenou DN a tím se získají náklady na incident. Proměnná „Spent time“ je celkový vynaložený čas na řešení incidentu. V práci se také setkáte s názvem úsilí (anglicky *Effort*), což bude značit právě tuto proměnnou. Ukázka dat je v Příloze A.

8.3 Identifikace chybějících a nekonzistentních dat

Jako první analýza, která se provede na připravených datech, je identifikace chybějících a nekonzistentních dat, protože chyby v datech mohou vést k nepřesným výsledkům dalších analýz, což může mít vážné důsledky pro rozhodování na základě těchto výsledků. Příčina chybějících

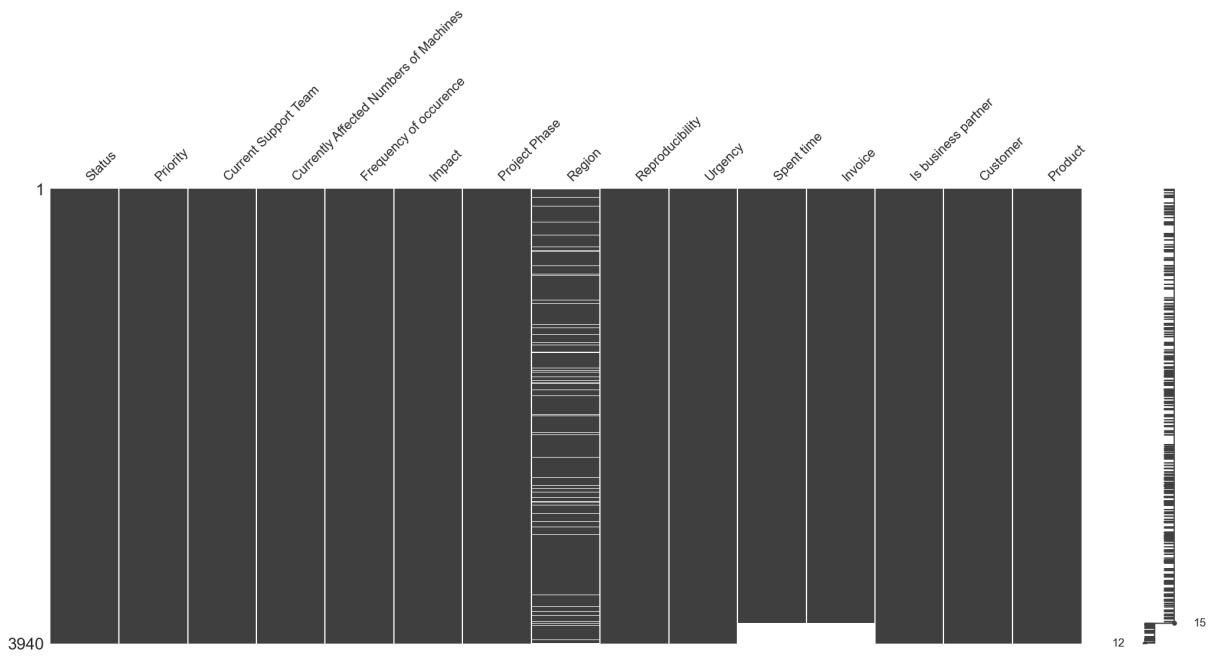
dat může být způsobena různými faktory, jako je jejich nepovinnost, systémová chyba nebo lidská chyba při jejich zadávání.

Nejdříve budou z modelu vyřazeny proměnné, které nebudou použity ve statistických metodách, ale budou použity až v aplikaci. Jedná se o proměnné „Product“ a „Customer“, které s analýzou nákladů v modelu nesouvisí a budou použity pro analýzu rozdílu úsilí mezi predikovanými a skutečnými hodnotami.

8.3.1 Chybějící data

Pro celý soubor byl vytvořen graf chybějících hodnot, který je zobrazen na obrázku 13. Chybějící hodnoty jsou vyznačené bílou barvou.

Obrázek 13: Graf chybějících hodnot



Zdroj: Vlastní zpracování, 2023

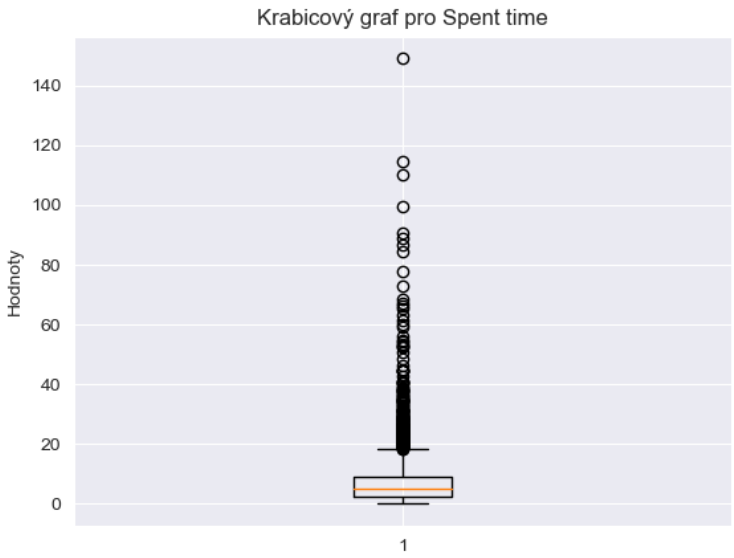
Ačkoliv je vidět, že ve sloupci „Status“ žádná hodnota nechybí, je nutné zde vyfiltrovat data. Jelikož cílem analýzy je predikce proměnné „Spent time“, vyřadíme ze sloupce všechny hodnoty jiné než „*Resolved*“ nebo „*Canceled*“. Vyřazené hodnoty značí stav incidentu, který ještě nebyl dokončen a je rozpracovaný — tedy výsledný čas není konečný a bude se ještě měnit. Dále vidíme chybějící hodnoty ve sloupci „Region“, které jsou „*NA*“, to je však způsobeno algoritmem, který hledá prázdné hodnoty, jako jsou např. *null*, *NaN*, *NA* atd. Proto hodnoty pouze nahradíme jiným názvem — „*NOA*“. Nakonec zbývají chybějící hodnoty ve sloupci „Spent time“, kde budou záznamy s chybějícími hodnotami ve sloupci jednoduše z modelu odstraněny,

celkem zbude k analýze 3 579 záznamů. Příčinou absence dat je neposkytnutí časových údajů o trvání incidentu z druhého systému.

8.3.2 Odlehlé hodnoty

Dalším krokem analýzy je odstranění odlehlých a extrémních hodnot, které by mohly způsobovat nepřesnost v následujících analýzách. Na odlehlé či extrémní hodnoty jsou některé statistické metody citlivé. K nalezení těchto hodnot bude využita metoda IQR (= *Interquartile Range*), která je definována jako rozdíl mezi třetím a prvním kvartilem (Q3 a Q1) a vypočítání horních a dolních vnitřních hranic pro odlehlá data a horních a dolních vnějších hranic pro extrémní data. Použití IQR místo standardního rozptylu má několik výhod včetně té, že je méně citlivá na odlehlé hodnoty. Odlehlé či extrémní hodnoty má cenu řešit pouze u kardinálních proměnných. Krabicový graf pro proměnnou „Spent time“ je vyobrazen na obrázku 14.

Obrázek 14: Krabicový graf pro „Spent time“



Zdroj: Vlastní zpracování, 2023

Z obrázku 14 je vidět, že sloupec obsahuje odlehlé hodnoty. Celkem obsahoval 225 odlehlých a extrémních hodnot, které byly odstraněny (opět celé záznamy). Po odstranění hodnot zbylo celkově v souboru 3 354 záznamů pro další analýzu. Odstraněné záznamy byly ojedinělé případy incidentů, na které se věnoval nadstandardní čas, jednalo se o případy např. se závažnou chybou, která se musela delegovat přes více oddělení/dodavatele atd. V tabulce 2 jsou vyobrazeny vypočtené charakteristiky pro sloupec „Spent time“, na kterých se metoda IQR aplikovala.

Tabulka 2: Charakteristiky IQR

Charakteristika	Hodnota
25% kvantil	2,57
75% kvantil	8,95
IQR	6,38
Horní vnitřní hradba	18,52
Dolní vnitřní hradba	-6,99
Horní vnější hradba	28,09
Dolní vnější hradba	-16,57

Zdroj: Vlastní zpracování, 2023

8.4 Vizualizace dat

Tato podkapitola představí vztahy mezi proměnnými pomocí grafických metod, které pomohou přehledně a názorně zobrazit informace obsažené v datech a lépe porozumět jejich vztahům a charakteristikám. V Příloze B jsou zobrazeny sloupcové grafy popisující rozdělení jednotlivých kategorií u nezávislých nominálních proměnných.

Z vizualizace je vidět, že z poskytnutých záznamů je nejvíce vytížený tým „*team-pm-banking-sw-vcps*“. Z proměnné „Impact“ lze vyčíst, že pouze malé množství incidentů je „*Critical*“ a že zbytek kritický není. Dalo by se také konstatovat, že polovina incidentů chodí od obchodního partnera a polovina ne. Dominantní prioritou je priorita „*Medium*“, tedy středně závažné incidenty, které musí DN řešit. Největší část incidentů také chodí již z fáze produkce, kdy je software již nasazený u zákazníka. Nejvíce incidentů pochází z regionu „*EMEA*“, takže se dá předpokládat, že nejvíce zákazníků je právě z tohoto regionu.

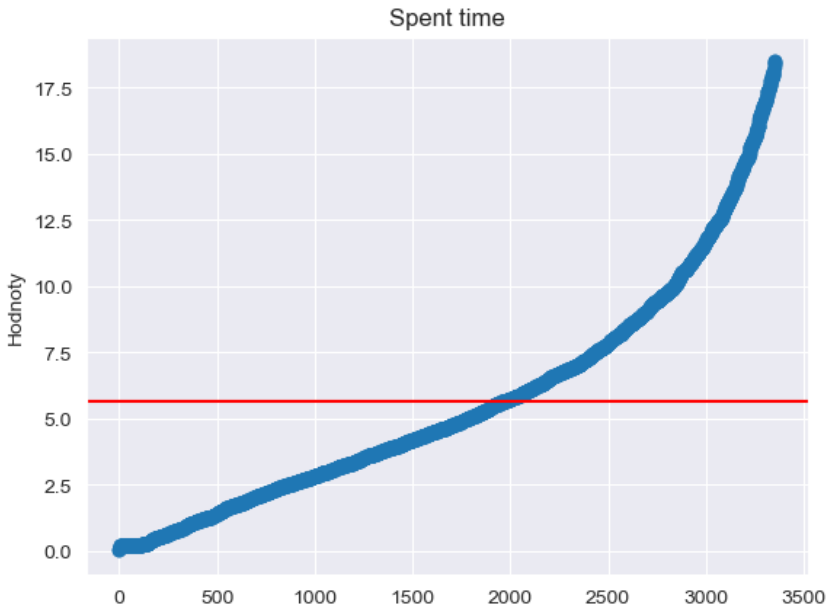
Co se týče proměnných „Reproducibility“ a „Status“, je u nich vidět podobnost, která může být způsobena tím, že proměnné jsou mezi sebou závislé. Tato hypotéza bude ověřena v kapitole Analýza závislostí. Pro kardinální proměnnou „Spent time“ se data promítla pomocí histogramu, který je zobrazen na obrázku 15. Červená přímka značí průměr hodnot zobrazované proměnné.

8.5 Popisná statistika

Popisná statistika je soubor metod a technik používaných k popisu a shrnutí souboru dat. Tyto metody zahrnují výpočet různých statistických ukazatelů, jako jsou průměr, medián, rozptyl a šikmost. Kategoriální data jsou zachycována pomocí jedno-, dvou- nebo vícerozměrných tabulek četností (kontingenční tabulky), relativních četností, nebo procent. (Hendl, 2015, s. 307)

Pro kardinální proměnnou byly vypočítány popisné statistiky, kterými lze charakterizovat a popsat soubor dat pomocí numerických charakteristik. Vypočtené statistiky ukazuje tabulka 3.

Obrázek 15: Seřazený histogram pro „Spent time“



Zdroj: Vlastní zpracování, 2023

Tabulka 3: Popisná statistika pro „Spent time“

Statistika	**Hodnota**
Počet	3354
Průměr	5,63
Směr. Od.	4,2
Minimum	0,03
Maximum	18,48
25% kvantil	2,44
50% kvantil	4,67
75% kvantil	7,97

Zdroj: Vlastní zpracování, 2023

Z popisných statistik lze usoudit například centrální tendenci, kdy statistiky průměr a medián naznačují, že hodnoty dat mají tendenci být kladně zešikmené (hodnoty se upínají k menším hodnotám). To také značí, že statistiky maximum a 75% kvantil, popisují, že 75 % hodnot je menších než hodnota 7,97, avšak maximum je 18,48. Tedy mezi intervalem 7,97 až 18,48 se nachází pouze 25 % hodnot.

Četnosti jednotlivých skupin v nominálních datech byly již nastíněny v minulé kapitole pomocí sloupcových grafů, avšak zde budou pro přehled do jednotlivých tabulek kromě četností také vypsány jejich přesné četnosti a procentuální rozložení v rámci proměnné, která jsou v Příloze C.

9 Analýza dat

Tato kapitola se zaobírá samotným modelováním statistických metod a ověřování hypotéz v rámci statistické analýzy, jejíž součástí je analýza korelací a ověření předpokladů modelů, interpretace výsledků modelů a porovnání jejich kvality na předem stanovených metrikách.

9.1 Analýza závislostí

Analýza závislostí byla provedena pro proměnné, u kterých bylo podezření na závislost. Vybrané podezřelé proměnné jsou „Reproducibility“ & „Status“. Jelikož se jedná o nominální kategorické proměnné, byl vybrán chí-kvadrát test (test dobré shody), který umožňuje testování závislosti mezi dvěma kategoriálními proměnnými. Testování je založeno na porovnání pozorovaných četností v jednotlivých buňkách kontingenční tabulky s očekávanými četnostmi, které jsou vypočteny za předpokladu, že neexistuje vztah mezi dvěma proměnnými. Pokud je rozdíl mezi pozorovanými a očekávanými četnostmi statisticky významný, znamená to, že mezi dvěma proměnnými existuje vztah, a jsou tedy závislé.

Formuluje se nulová a alternativní hypotéza:

H_0: Mezi proměnnými „Reproducibility“ a „Status“ neexistuje žádný vztah nebo závislost.
H_1: Mezi proměnnými „Reproducibility“ a „Status“ existuje vztah nebo závislost.

Pro testování závislosti byly vytvořeny kontingenční tabulky, které byly vstupem pro test dobré shody. Po provedení testu vyšla p-hodnota menší než zvolená α. Je zamítnuta nulová hypotézu na úrovni α a můžeme o proměnných tvrdit, že jsou mezi sebou závislé. Mezi proměnnými vyšla p-hodnota $6,87E-7$. Byly vypočítány statistické míry — **koeficient kontingence** pro měření podobnosti rozdělení hodnot mezi dvěma nominálními proměnnými (0 znamená, že proměnné jsou nezávislé, a 1, že jsou úplně závislé) a **Cramerovo V**, které měří síly vztahu mezi dvěma nominálními proměnnými (0 znamená úplnou nezávislost a 1 úplnou závislost). Z tabulky 4 lze podle zmíněných statistickým měr vyčíst, že data jsou mezi sebou téměř zcela **nezávislá**. Jde tedy pouze o náhodu, že četnosti hodnot v obou zmíněných proměnných jsou podobné.

Tabulka 4: Výsledky analýzy závislostí

Proměnná	Chí-kvadrát statistika	p-hodnota	Koeficient kontingence	Cramerovo V
Reproducibility & Status	24,65	6,87E-07	7,35E-03	8,57E-02

Zdroj: Vlastní zpracování, 2023

9.2 Ověření normality

U ověření normality bude prováděna analýza pouze pro kardinální proměnnou, která je v souboru pouze jedna — „Spent time“. Normalita dat bude ověřena pomocí (K-S) Lillieforsova testu a graficky se hypotéza ověří podle kvantilového (Q-Q) grafu, který porovnává rozdělení dat s teoretickým normálním rozdělením. K-S testuje nulovou hypotézu, že data pocházejí z normálního rozdělení. Pokud výsledek testu je statisticky významný, znamená to, že máme důkazy o odmítnutí nulové hypotézy a tedy o tom, že data pocházejí z jiného rozdělení než normálního.

Formuluje se nulová a alternativní hypotéza:

H_0: Proměnná „Spent time“ pochází z normálního rozdělení.
H_1: Proměnná „Spent time“ nepochází z normálního rozdělení.

Po provedení testu vyšla p-hodnota menší než zvolená α. Konkrétně vyšla 9,99E-4. Je možné prohlásit, že proměnná nepochází z normálního rozdělení, a tudíž p-hodnota nulovou hypotézu **zamítá** na úrovni α podle K-S testu. Na obrázku 16 je Q-Q graf, kde jsou vykresleny teoretické a skutečné hodnoty proměnné.

Obrázek 16: Q-Q graf pro „Spent time“

Zdroj: Vlastní zpracování, 2023

Po tomto kroku je nutné nominální kategorická data nahradit číselnými hodnotami pro další analýzu, jelikož software, ve kterém je prováděna statistika, pracuje pouze s číselnými hodnotami. Autor se rozhodl pro metodu tzv. *label encoding* neboli každá kategorie v daném sloupci je nahrazena vlastní číselnou hodnotou. Kompletní přehled spojovacích tabulek je v Příloze D.

9.3 Regresní analýza

Následující kapitola by pokračovala provedením regresní analýzy, avšak z předpokladů regresní analýzy se musí model poupravit, jelikož předpoklady nejsou splněny:

- Lineární vztah mezi nezávislými a závislou proměnnou — To znamená, že změny v nezávislých proměnných mají přímý lineární vliv na závislou proměnnou, avšak v modelu se vyskytuje příliš mnoho nezávislých nominálních kategoriálních dat, která jsou založená na diskrétních, neuspořádaných kategoriích, jež nelze promítat na lineární škále. Tudíž tento předpoklad může být **zamítnut**.
- Normalita závislých proměnných — Tento předpoklad již autor ověřil v kapitole Ověření normality, kde zamítl H_0, neboť proměnná „Spent time“ nepochází z normálního rozdělení, tyto hodnoty závislé proměnné nejsou rovnoměrně rozloženy kolem střední hodnoty, a tudíž autor tento předpoklad může také **zamítnout**.

Pro nesplnění předpokladů regresní analýzy se autor rozhodl pozměnit model a dále pokračovat v diskriminační analýze, kdy závislou proměnnou „Spent time“ rozdělí do několika intervalů tak, aby se z ní stala kategorická proměnná, která dále bude moc být klasifikována právě díky zmíněné metodě.

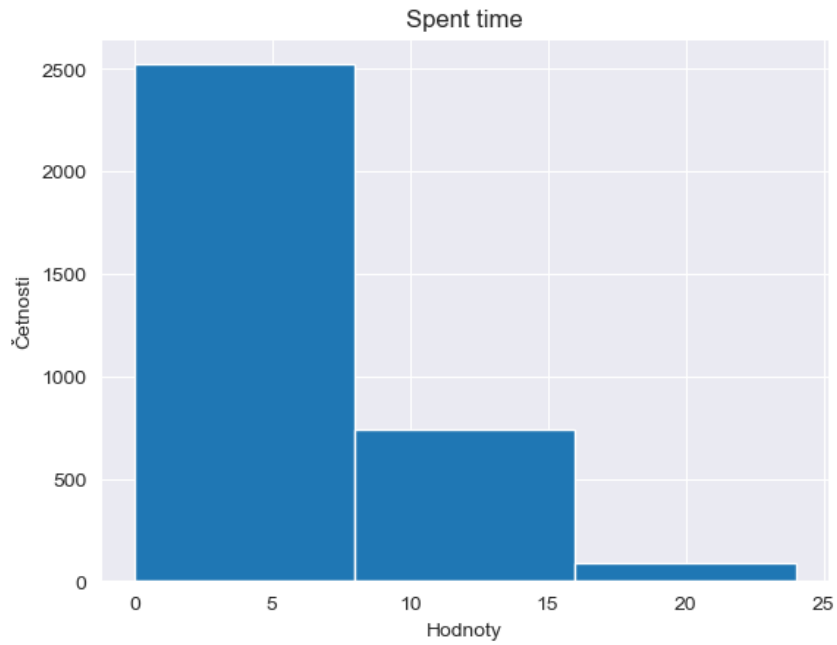
9.4 Změna modelu

Jak bylo zmíněno, je potřeba převést kardinální závislou proměnnou „Spent time“ na nominální, tedy na jednotlivé intervaly (kategorie), které pak půjdou odhadovat na základě některého diskriminačního modelu.

Z grafu 17 rozložení hodnot proměnné „Spent time“ rozděleného na intervaly po 8 jednotkách je vidět, že data se dají rozdělit podle člověkodnů (MD = *Manday*). „Člověkoden je pracovní čas jedné osoby odpovídající jednomu pracovnímu dni, tedy typicky 8 hodin.“ (Management Mania, 2018). Autor rozdělení diskutoval s vedoucím z DN — Ing. Davidem Kriváneknem (osobní komunikace, 6. 2. 2023) a bylo mu řešeno, že takto rozdělená data jsou přijatelná a uchopitelná, jelikož pro oddělení M&S je podstatné to, jestli strávený čas na incidentu bude do 1 MD nebo ho překročí, což je z tohoto rozdělení patrné.

Tímto rozdělením dojde k tzv. diskreditaci proměnné „Spent time“. Pokud bude výsledný model kvalitní (bude predikovat správně), může se predikovat alespoň v řádech pracovních dní.

Obrázek 17: Histogram diskreditace proměnné „Spent time“



Zdroj: Vlastní zpracování, 2023

9.5 Korespondenční analýza

Jelikož se v modelu nacházejí nezávislé nominální proměnné, bude provedena korespondenční analýza, která slouží k analýze vztahů mezi dvěma nebo více kategoriálními proměnnými. Jejím cílem je identifikovat vztahy mezi kategoriemi jedné proměnné a kategoriemi druhé proměnné a zobrazit je v nízkorozměrném prostoru při maximálním zachování informace. Výsledkem analýzy je tzv. korespondenční mapa představující osy redukovaného souřadného systému, ve kterém jsou graficky zobrazeny jednotlivé kategorie obou proměnných za pomoci latentních proměnných, jež vyjadřují asociaci (vzdálenost řádkových a sloupcových profilů) ve stejné dimenzi. V Příloze E se nacházejí korespondenční mapy mezi nezávislými proměnnými a závislou „Spent time“. (Komenda, 2014; Poláčková & Jindrová, 2010)

Z vytvořených korespondenčních map můžeme interpretovat následující výsledky:

(a) Lze vypozorovat shluk kolem hodnot závislé proměnné „*1*“ (0-1 MD) a „*2*“ (1-2 MD), zde hodnoty nezávislé proměnné přispívají ke klasifikaci do těchto kategorií. Avšak vidíme zde osamocené hodnoty „*team-pm-banking-sq-proakt*“ a „*3*“ (2-3 MD).

(b) Pokud incident ovlivňuje menší počet strojů „*1*“ a „*10-100*“, bude incident spadat do kategorie „*1*“. „*2-10*“ počet strojů je na pomezí kategorií „*1*“ a „*2*“. Vetší počty „*100-1000*“ a „*more than 1000*“ naznačují spíše kategorii „*2*“. Kategorie „*3*“ závislé proměnné je opět osamocena.

(c) Graf poukazuje na osamocenou kategorii incidentu „*3*“, ke kterému nepřispívá žádná z hodnot nezávislé proměnné. Z analýzy také vyplývá, že pokud jsou problémy v incidentu více frekventované, přispívá to k navýšení celkového úsilí na řešení daného incidentu.

(d) Pokud je hodnota nezávislé proměnné „*Critical*“, bude to ovlivňovat celkovou dobu nutnou na vyřešení incidentu, jelikož jí bude navyšovat.

(e) Hodnota nezávislé proměnné „*yes*“ přispívá k zařazení do skupiny závislé proměnné „*2*“ a hodnota „*no*“ rovněž do „*2*“. Kategorie „*3*“ je oddělená od ostatních.

(f) Nejvíce se odlišuje hodnota nezávislé proměnné „*Highest*“, která nepřispívá ani k jedné kategorii závislé proměnné. Avšak hodnota „*Medium*“ přispívá spíše do „*2*“ a „*3*“, hodnoty „*High*“ a „*Low*“ do „*1*“.

(g) Lze vypozorovat tři skupiny. První je shluk kolem kategorie závislé proměnné „*1*“ a hodnot „*Internal QA*“, „*Production*“ a „*UAT*“ nezávislé proměnné. Druhou je shluk kolem kategorie „*2*“ a k ní přidružených hodnot „*Certification*“, „*SIT*“ a „*Pilot*“ nezávislé proměnné. Poslední skupinou je kolem kategorie závislé proměnné „*3*“, ke které nepřispívá žádná hodnota za nezávislé proměnné.

(h) V tomto grafu lze také vypozorovat tři skupiny. První shluk je kolem kategorie závislé proměnné „*1*“ a hodnot „*NOA*“, „*EMEA*“ a „*APAC*“ nezávislé proměnné. Z analýzy vyplývá, že pokud se jedná o incident, který pochází z oblasti Latinské Ameriky, bude jeho trvání delší.

(i) Graf zobrazuje shluky kolem kategorií závislé proměnné „*1*“ a „*2*“, kde hodnota „*Highest*“ nezávislé proměnné přispívá ke klasifikaci do kategorie „*2*“ a hodnoty „*High*“ a „*Medium*“ do kategorie „*1*“. Osamocená je hodnota „*Low*“, která výrazně nikam nepřispívá, a kategorie „*3*“, ke které žádná hodnota nespadá.

(j) Pokud je daný problém reprodukovatelný, přispívá k navýšení úsilí daného incidentu, avšak kategorie „*3*“ závislé proměnné je opět poměrně osamocena.

(k) Pokud je incident vyřešen, je zřejmé, že úsilí incidentu bude spadat do kategorie „*1*“, v opačném případě je nutné věnovat větší úsilí zkoumání příčin incidentu, který se ve finále zamítne. Dále je řešené, že hodnota „*Canceled*“ přispívá ke zvýšení doby úsilí na daném incidentu, jelikož je blíže kategoriím „*2*“ a „*3*“.

9.6 Diskriminační analýza

Diskriminační analýza slouží k rozlišení (diskriminaci) proměnné, která má konečný počet tříd. Jedná se o metodu predikce s učitelem, kdy se pro vytvoření diskriminační (klasifikační) funkce a ověření její účinnosti použijí trénovací a testovací data. Tato data jsou již označena (mají již

přidělenou třídu) z jednoho souboru, který se rozdělil na tyto dvě množiny. (Gangur & Svoboda, 2021)

Cílem analýzy bude nejen nalézt kombinaci nezávislých proměnných, která nejlépe rozděluje jednotlivé třídy v závislé proměnné, ale také najít diskriminanty (nezávislé proměnné), které nejlépe oddělují jednotlivé třídy. Do modelu budou vstupovat diskriminanty a závislá proměnná „Spent time“, která je nyní již také kategoriální.

Definuje se nulová a alternativní hypotéza:

H_0: Neexistuje statisticky významný rozdíl mezi skupinami vzhledem k danému souboru proměnných.

H_1: Existuje statisticky významný rozdíl mezi skupinami vzhledem k danému souboru proměnných.

9.6.1 Lineární diskriminační analýza

Lineární diskriminační analýza (LDA = *Linear Discriminant Analysis*) používá k nalezení lineární kombinace funkcí, která dokáže nejlépe oddělit několik tříd v datech. Metoda se snaží minimalizovat vnitřní variabilitu v rámci tříd a maximalizovat variabilitu mezi třídami.

Byla zvolena kroková analýza (anglicky *Forwardstepwise*), která umožňuje přidávat proměnné do modelu podle jejich významu, v tomto případě od nejvíce významné po nejméně významnou. Bylo navoleno celkem 11 kroků (celkový počet nezávislých proměnných).

Obrázek 18: Výsledky LDA

N=3354	Výsledky diskriminační funkční analýzy (Sheet1 v prepared-data-v8_cat) Počet prom. v modelu: 11; grupovací: Spent_time (3 skup) Wilk. lambda: ,98059 přibliž F (22,6682)=2,9909 p< ,0000					
	Wilk. První	Parc. První	F na vyj 2,3341	p-hodnot	Toler.	1-toler. R^2
Status	0,981582	0,998992	1,685713	0,185470	0,964987	0,035013
Priority	0,983797	0,996743	5,458602	0,004298	0,954336	0,045664
Current_Support_Team	0,981438	0,999138	1,440772	0,236892	0,976994	0,023006
Currently_Affected_Numbers_of_Machines	0,982869	0,997684	3,877438	0,020797	0,905967	0,094033
Frequency_of_occurence	0,984466	0,996066	6,597678	0,001381	0,719940	0,280060
Impact	0,984739	0,995790	7,063066	0,000869	0,964801	0,035199
Project_Phase	0,980973	0,999612	0,648596	0,522845	0,964694	0,035306
Region	0,981559	0,999016	1,646084	0,192960	0,898633	0,101367
Reproducibility	0,980907	0,999679	0,535883	0,585203	0,825916	0,174084
Urgency	0,981275	0,999305	1,161778	0,313056	0,917464	0,082536
Is_business_partner	0,983511	0,997033	4,971906	0,006981	0,764351	0,235649

Zdroj: Vlastní zpracování s využitím Statistica, 2023

Z výsledků diskriminační analýzy (obrázek 18) je vidět, že kvalita modelu W (= Wilcova lambda) se blíží téměř k žádné diskriminaci (W = 1), to znamená, že model není schopen dobře kvalifikovat diskriminanty do kategorií závislé proměnné. Vypočtená p-hodnota je menší

než zvolená α, a proto je nutné nulovou hypotézu na této úrovni α **zamítnout**. Zamítnutí nulové hypotézy neznamená nutně, že jsou všechny skupiny statisticky významně odlišné. Červené proměnné jsou diskriminátory, které statisticky přispívají k celkovému vlivu zařazení do správné třídy. Celkem je významných 5 proměnných, ostatní proměnné se mohou z modelu vynechat, jelikož jejich statistická významnost by výsledky testu nijak výrazně neovlivnila. Největší vliv má proměnná „Is business partner“ a nejmenší „Project phase“.

Analýza byla provedena také v programovacím jazyce Python, který bude dále v práci použit pro tvorbu aplikace a bude více popsán v kapitole Aplikace. V rámci psaného skriptu byl rozdělen trénovací a testovací soubor na poměr 7:3 a přidal se parametr pro náhodné zamíchání dat před rozdělením na tyto dvě množiny. Pro 10 000 iterací se ukázala, jako nejvíce přesná 7 103. náhodná iterace, která bude použita také pro další analýzy v této kapitole.

Následující ukázka kódu představuje rozdělení dat na trénovací a testovací:

```
# Priprava vstupnich a vystupnich dat
y = data[selected_cols]     # Nezavisle promenne
X = data['Spent_time']     # Zavisla promenne

# Rozdeleni data na trenovaci a testovaci
X_train, X_test, y_train, y_test = train_test_split(y, X,
test_size=0.3, random_state=7103)
```

Následující ukázka kódu představuje definici modelu lineární diskriminační analýzy v modelu:

```
# Inicializace LDA modelu
lda = LinearDiscriminantAnalysis(n_components=2)

# Trenovani LDA modelu
lda.fit(X_train, y_train)

# Predikce ciloveho atributu
y_pred = lda.predict(X_test)

# Vyhodnoceni presnosti modelu
accuracy = accuracy_score(y_test, y_pred)
```

Podle definovaného modelu a rozdělení testovacích dat vyšla přesnost modelu 74,578 %. Přesnost (anglicky *Accuracy*) klasifikační metody je metrika pro vyhodnocování výkonnosti klasifikačních modelů. Značí se jako podíl počtu správně klasifikovaných pozorování k celkovému počtu pozorování v testovací množině. V práci se bude používat jako hlavní metrika.

9.6.2 Kvadratická diskriminační analýza

Jazyk Python umožňuje také provést kvadratickou diskriminační analýzu (QDA = *Quadratic Discriminant Analysis*). Od LDA se liší tím, že se snaží nalézt kvadratické kombinace funkcí, které by rozlišily jednotlivé třídy místo lineárních. Umožňuje tak lépe zachycovat nelineární vztahy mezi prediktory a třídami.

Model QDA je v kódu definován takto:

```
# Inicializace LDA modelu
qda = QuadraticDiscriminantAnalysis(n_components=2)

# Trenovani LDA modelu
qda.fit(X_train, y_train)

# Predikce ciloveho atributu
y_pred = qda.predict(X_test)

# Vyhodnoceni presnosti modelu
accuracy = accuracy_score(y_test, y_pred)
```

Podle definovaného modelu a rozdělení testovacích dat vyšla přesnost modelu 73,6842 %. Jedná se tedy o menší přesnost než u LDA, to avšak ještě neznamená, že by model hůře klasifikoval, podrobnější porovnání modelů se nachází v kapitole Interpretace výsledků.

9.7 Metody strojového učení

V této podkapitole budou představeny metody strojového učení zaměřené na klasifikaci.

9.7.1 Náhodný les

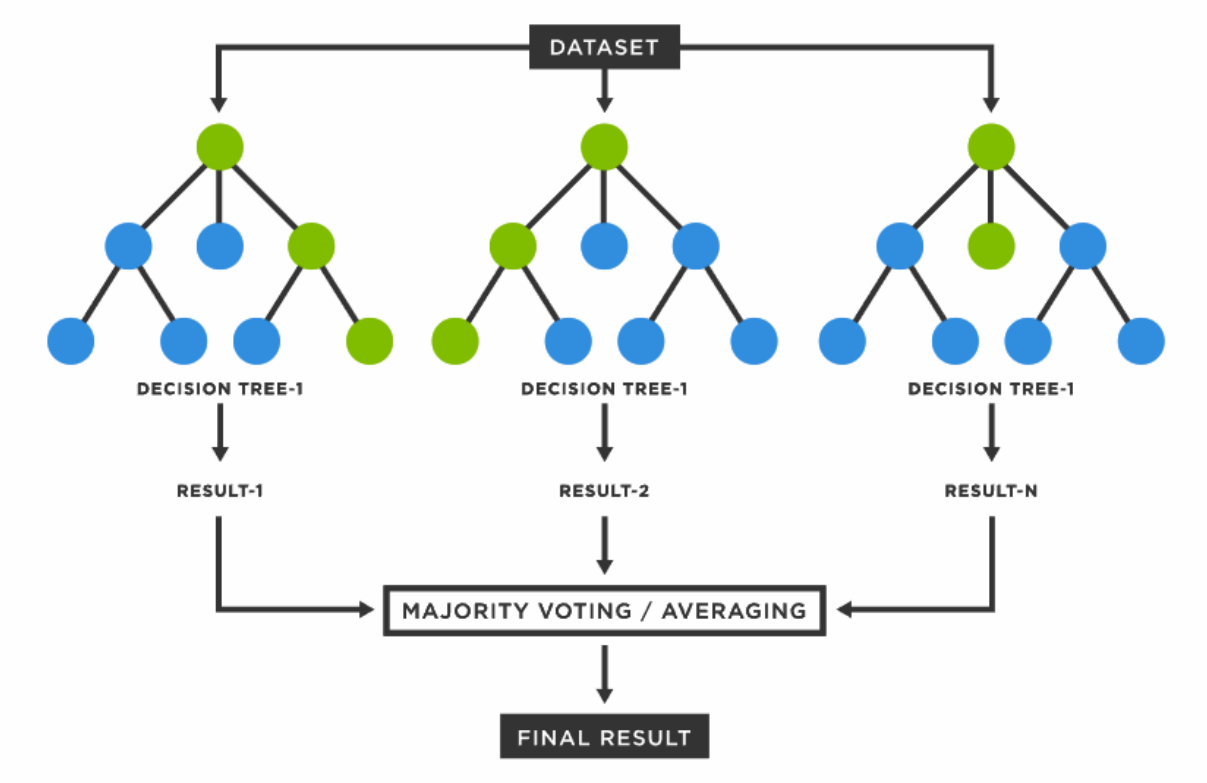
Metoda náhodných lesů (RF = *Random Forrest*), je založena na rozhodovacích (klasifikačních) stromech. Kombinuje rozhodovací stromy do většího modelu, který poskytuje lepší výkon na náhodně vybraných podmnožinách dat a náhodně vybírá podmínky pro každý rozhodovací uzel v každém stromu. Tím se snižuje pravděpodobnost přeučení (anglicky *Overfitting*) na trénovacích datech a zvyšuje se obecnost (generalizace) modelu. Každý strom pak dává svou klasifikační nebo regresní predikci a výsledná predikce je zprůměrována přes všechny stromy. (Klaschka & Kotrč, 2004)

Popis principu fungování

Uvnitř lesu se používají binární stromy typu **CART** (= *Classification and Regression Trees*). Rozdělení dat probíhá na základě jedné proměnné a její hodnoty tak, aby byla minimalizována chybovost klasifikace. Rozdělení dat je uvnitř stromu (uzlu) založeno na podmínce, když je

podmínka splněna, pokračuje „pravda“ větví, jinak pokračuje do „nepravda“ větví. Schéma obecné struktury náhodného stromu ukazuje obrázek 19.

Obrázek 19: Obecné schéma RF



Zdroj: (TIBCO Software, 2023)

Postup výpočtu metody náhodných lesů je následující:

1. Nejdříve je vybrána náhodná podmnožina z trénovací sady (tzv. *Bootstrap Sample*), která bude použita pro vytvoření jednoho stromu.

2. Při každém dělení uzlu stromu se vybere pouze náhodná podmnožina proměnných, které jsou dostupné pro toto dělení. Tento krok zajišťuje, že stromy budou různorodé.

3. S použitím výše zmíněných náhodných výběrů dat a proměnných se vytvoří rozhodovací strom. Tento strom se vytváří dělením uzlů na základě některé hodnoty z proměnných a hledání nejlepšího dělícího bodu.

4. Tento postup se opakuje mnohokrát, aby se vytvořily různé stromy.

5. Výsledné rozhodnutí je založeno na hlasování (pro klasifikaci), tedy na tom, jak často se určitá třída vyskytuje v rozhodnutích jednotlivých stromů.

6. Pro nová data se provede predikce, kdy nejprve se data procházejí jednotlivými stromy, aby byly získány výsledky pro každý strom. Následně se použije hlasování pro získání konečné predikce. (Kovalenko a kol., 2022)

Výběr parametrů

Výsledky RF modelu ovlivňují nejvíce dva parametry, a to „*n_estimators*“ a „*random_state*“. První zmíněný parametr určuje, kolik stromů bude vytvořeno v lesu a jak moc bude tento les komplexní. Zvýšení hodnoty může vést k lepší schopnosti modelovat složité vzory v datech. Nicméně vyšší hodnota také může vést k delšímu času tréninku a vyššímu výpočetnímu výkonu. Druhý parametr určuje semínko pro generování náhodných čísel v algoritmu. RF metoda používá náhodné výběry příznaků a dat pro každý strom v lesu. Tím, že nastavíme stejné semínko, zajistíme, že pro každý běh algoritmu bude použit stejný náhodný výběr, což nám umožní získat reprodukovatelné výsledky, což autor využije v aplikaci.

Pro zjištění nejlepšího parametru „*n_estimators*“ bylo uskutečněno 100 běhů s krokem 25. Ze zvolených metrik bylo pro výběr nejlepšího modelu použito F-skóre, které bere v úvahu jak přesnost (anglicky *Precision*), tak úplnost (anglicky *Recall*) a je vhodné pro nevyvážená data. F-skóre je harmonický průměr zmíněných metrik. Metriky přesnost a úplnost budou více vysvětleny v kapitole Interpretace výsledků. Protože existují třídy s různým počtem pozorování, bude pro výpočet F-skóre použita průměrná hodnota (anglicky *Macro Average*) pro všechny třídy, což zaručí, že menší třídy budou mít stejný vliv na výsledek jako větší třídy. Tabulka 5 zobrazuje vybrané počty rozhodovacích stromů a jejich metriky.

Pro nalezení druhého parametru bylo uskutečněno 1 000 běhů s nastavením parametru „*n_estimators*“ na hodnotu 151. Parametr „*random_state*“ s největší hodnotou F-skóre (0,3578) vyšel 597.

Následující ukázka kódu představuje definici modelu náhodného lesa:

Tabulka 5: Porovnání počtu rozhodovacích stromů v modelu

Počet rozhodovacích stromů	Průměrná přesnost	Průměrná úplnost	Průměrné F-skóre
1	0,3496	0,347	0,346
26	0,3509	0,3445	0,3368
51	0,3534	0,3462	0,3383
101	0,3671	0,3512	0,3439
151	0,3696	0,3537	0,3472
201	0,3523	0,3441	0,3344
501	0,3417	0,339	0,3283
1001	0,3437	0,3399	0,3292
2476	0,3414	0,3391	0,3291

Zdroj: Vlastní zpracování, 2023

```
# Definice modelu
rf_model = RandomForestClassifier(n_estimators=151, random_state=597)

# Trenovani modelu
rf_model.fit(X_train, y_train)

# Predpoved na testovacich datech
y_pred = rf_model.predict(X_test)

# Vypocet presnosti modelu
accuracy = accuracy_score(y_test, y_pred)
```

Na základě trénovacího souboru a vypočítaných parametrů vyšla nejlepší přesnost modelu 70,3078 %.

9.7.2 Neuronová síť

Pposlední metoda je neuronová síť (NN = *Neural Network*), kde základním stavebním kamenem je neuron, název byl převzat z biologie. Neuronová síť je orientovaný graf, kde neurony (uzly) jsou vzájemně propojeny spoji s ohodnocenými vahami. „Takovéto propojení a schopnost tyto váhy adaptovat (učit se) na základě trénovacích vzorů v datech dává neuronové síti nové široké možnosti v oblasti analýzy dat.“ (StatSoft, 2013).

Neuron se dá popsat matematiky následovně:

$$y = f(\sum_{i=1}^{n} x_i \times w_i - \theta) \tag{3}$$

Zdroj: (Wikipedie, 2023d; StatSoft, 2013), zpracováno autorem

kde: y ... hodnota výstupu,
n ... celkový počet vstupů,
x_i ... hodnota na i-tém vstupu,
w_i ... váha i-tého vstupu,
θ ... prahová hodnota.

Prahová hodnota určuje, zda bude neuron aktivován, či nikoliv. Pokud je výsledná hodnota po aplikaci aktivační funkce vyšší než prahová hodnota, neuron bude aktivován a výstup z něj bude předán dalším neuronům v síti. Pokud je výsledná hodnota nižší než prahová hodnota, neuron nebude aktivován a jeho výstup bude nulový.

Samotný neuron není schopen vykonávat lepší výsledky než např. regresní analýza, proto se neurony shlukují do vrstev. Neuronová síť se skládá ze vstupní vrstvy, jedné až n skrytých vrstev

a vrstvy výstupní. Neuronová síť díky vazbám a přenastavováním vah během učení dokáže „[...] najít i složitější a nelineární vztahy, na druhou stranu je ale pravda, že ze získané neuronové sítě nikdy nebudeme schopni získat interpretaci, proč to u konkrétního pozorování dopadlo, jak to dopadlo. Je to tedy metoda typu „black box“ – nejsme schopni jednoduše interpretovat výsledky či získat jednoduchý předpis závislosti mezi závislou a nezávislými proměnnými.“ (StatSoft, 2013).

Výběr sítě

V této práci se autor rozhodl použít neuronové sítě typu MLP (= *Multilayer Perceptron*), neboť se jedná o jeden z nejčastěji používaných typů neuronových sítí pro klasifikaci. MLP sítě jsou vícevrstvé sítě, které se skládají z jedné vstupní vrstvy, jedné nebo více skrytých vrstev a jedné výstupní vrstvy. Pro výběr správného modelu budou do modelu postupně přidávány skryté vrstvy vždy s dvojnásobným počtem neuronů. Pro zvolení nejlepšího modelu byla využita metrika F-skóre, podobně jako u RF. V tabulce 6 jsou uvedeny všechny navržené modely neuronových sítí.

Tabulka 6: Modely neuronových sítí

Model	Průměrná přesnost	Průměrná úplnost	Průměrné F-skóre
MLP-64-3	0,25	0,33	0,28
MLP-128-64-3	0,34	0,33	0,3
MLP-256-128-64-3	0,35	0,34	0,32
MLP-512-256-128-64-3	0,32	0,33	0,3
MLP-1024-512-256-128-64-3	0,3	0,33	0,29

Zdroj: Vlastní zpracování, 2023

Z tabulky vyplývá, že podle zvolené metriky nejlépe klasifikuje model „MLP-256-128-64-3“. Vybraný model bude popsán v následující podkapitole. Z výsledků si také můžeme všimnout, že od jistého bodu modely klasifikovaly se stále menší přesností, pokud se přidalo příliš skrytých vrstev.

Popis sítě

Model zvolené neuronové sítě má celkem čtyři vrstvy. Vstupní vrstva má dimenzi 11 (odpovídá vstupu proměnných). To znamená, že tato vrstva má 11 vstupů a 256 výstupů. Druhá vrstva obsahuje 128 neuronů a využívá výstupy z předchozí vrstvy jako své vstupy. Třetí vrstva má 64 neuronů a opět využívá výstupy z předchozí vrstvy. Výstupní vrstva je klasifikační vrstva a obsahuje 3 neurony, což odpovídá třem kategoriím, do kterých se vstupní data klasifikují.

První (vstupní) vrstva a skryté vrstvy obsahují aktivační funkci **ReLU** (= *Rectified Linear Unit*), která pro vstupy menší nebo rovno nula vrací nulu a pro vstupy větší než nula vrací hodnotu vstupu. Neboli:

$$f(x) = \max(0, x) \tag{4}$$

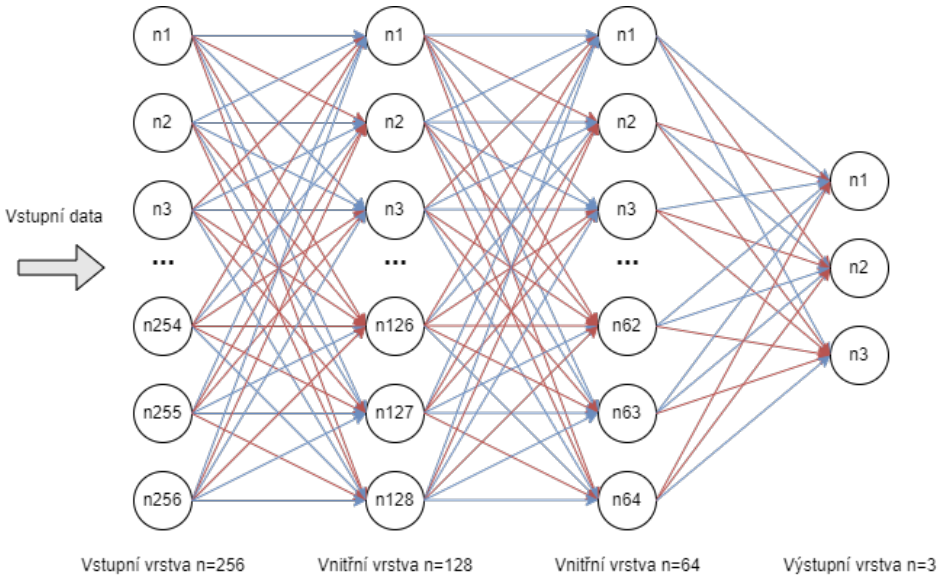
Zdroj: Vlastní zpracování, 2023

kde: f(x) ... hodnota na výstupu neuronu,
x ... vstupní hodnota neuronu.

Výstupy poslední vrstvy jsou transformovány pomocí **softmax** aktivační funkce, která zajišťuje, že všechny výstupy jsou v rozmezí 0 a 1 a součet všech výstupů je 1. Tento proces zajišťuje pravděpodobnostní interpretaci výstupů a umožňuje vybrat kategorii s nejvyšší pravděpodobností.

Kompilace modelu používá kategoriální **křížovou entropii** (anglicky *Cross Entropy*) jako ztrátovou funkci. Tato funkce je často používána pro klasifikační úlohy, kde cílem je minimalizovat rozdíl mezi pravdivými a predikovanými výstupy. Optimalizační funkcí je **Adam**, což je adaptivní metoda gradientního sestupu, která umožňuje rychlé a účinné trénování sítě. Metrikou pro hodnocení výkonu modelu je opět přesnost pro pozdější porovnání kvality metod.

Obrázek 20: Schéma navržené neuronové sítě



Zdroj: Vlastní zpracování, 2023

Barva spojů na obrázku 20 je náhodně vygenerovaná, ale svým způsobem vyjadřuje, jak jednotlivé váhy přispívají k vyhodnocování neuronů, kde červená barva je ve prospěch aktivace neuronu a modrá pro opak.

Následující ukázka kódu představuje definici modelu neuronové sítě v modelu:

```
# Definovani modelu neuronove site
```

```
model = Sequential()
model.add(Dense(256, input_dim=11, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))  # 3 kategorie a
                                           # softmax aktivace

# Kompilace modelu
nn_model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Nauceni modelu na trenovacich datech
nn_model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

# Vyhodnoceni modelu na testovacich datech
y_pred = nn_model.predict(X_test)
```

Na testovacích datech vyšla přesnost modelu $71,8967$ %. Vyhodnocení navržených metod klasifikace je více popsáno v kapitole Interpretace výsledků.

10 Aplikace

Autor práce se rozhodl demonstrovat výsledky klasifikačních modelů pomocí webové aplikace. Pro vývoj na tuto platformu se rozhodl z toho důvodu, že současný vývoj umožňuje tvorbu takových webových aplikací, které jsou snadno škálovatelné a přenositelné mezi různými platformami (iOS/Android, Linux, Windows atd.). Jádro aplikace však zůstává stejné, backend aplikace běží na serveru a komunikuje s klientem (frontendem) pomocí veřejné nebo interní sítě.

10.1 Technologický stack

Frontend aplikace je napsaný v jazyce **Typescript**, jenž je nadstavbou jazyka Javascript. Mezi hlavní rozdíly patří statická typová kontrola, která pomáhá vývojářům identifikovat chyby v kódu ještě před během programu. Právě Typescript podporuje framework **Angular**, který je zaměřený na tvorbu moderních webových aplikací. Angular se zaměřuje na rozdělení aplikace do oddělených komponent, které jsou znovupoužitelné a mohou být snadno modifikovány. Každá komponenta má vlastní šablonu, která definuje její vzhled a chování. Komponenty jsou poté sestaveny do hierarchické struktury, která tvoří celou aplikaci. Jako každá webová aplikace, tak vzhled a struktura je psaná v jazyce **HTML 5** a **CSS**, který je v rámci frameworku rozšířen o předprocesor kaskádových stylů **Sass**, což umožňuje programátorům psát CSS kód s využitím pokročilejších funkcí, než které jsou k dispozici v základním CSS.

Backend aplikace, který běží na serveru, je napsaný v jazyce **Python** z toho důvodu, že i statistická analýza byla psána v tomto jazyce a instance naučených modelů jsou zde uloženy. Python je jeden z nejpopulárnějších jazyků pro vývoj backendových aplikací, umožňuje jednoduchost syntaxe a čitelnost kódu, modularitu a širokou škálu nabízených knihoven a frameworků. Pro komunikaci mezi oběma částmi aplikace je využit framework **FastAPI**, který umožňuje psát tzv. API (= *Application Programming Interface*) — rozhraní mezi různými aplikacemi, jež umožňuje komunikaci a výměnu dat. Pokud se jedná o analýzu dat, jsou zde použité patřičné statistické knihovny. Při psaní statistických tříd autor vycházel ze zdroje (McKinney, 2018).

10.2 Popis aplikace

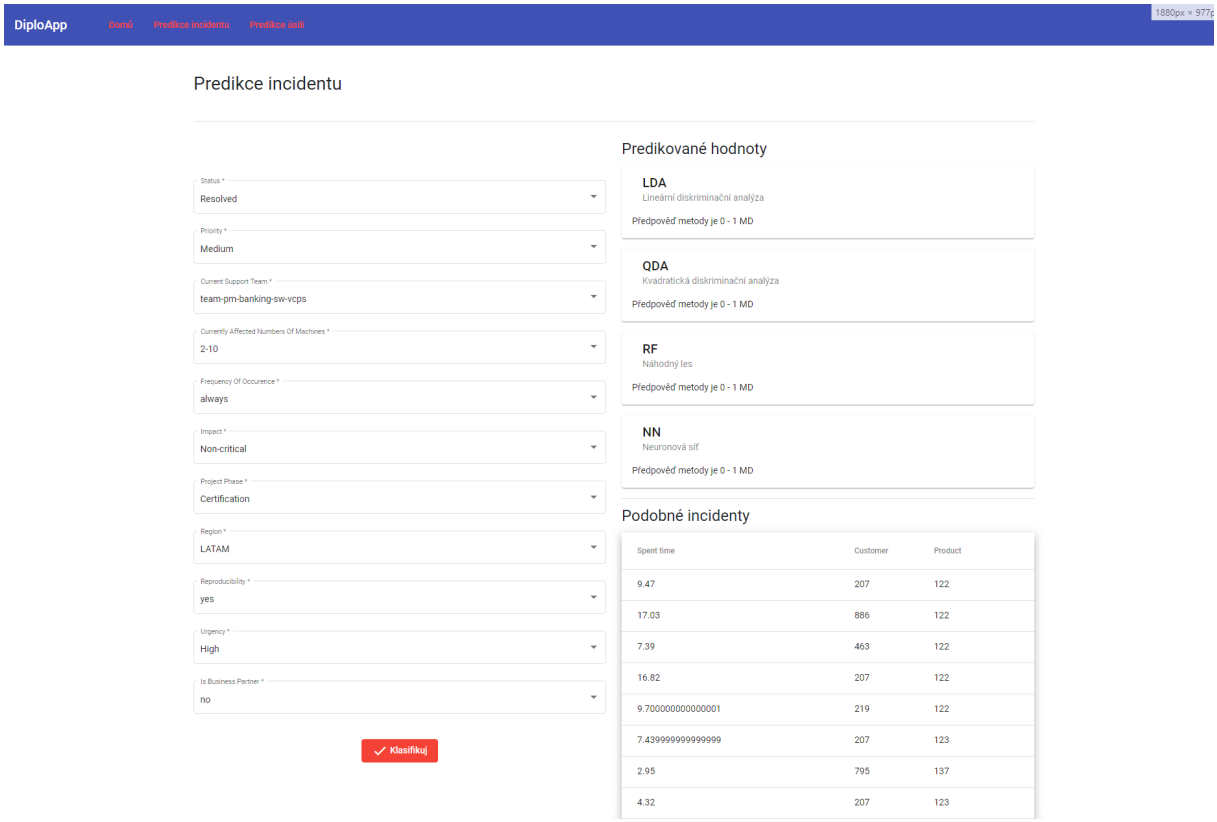
10.2.1 Komponenty

Aplikace se skládá celkem ze dvou komponent, které umožňují různou funkcionalitu aplikace. Dané komponenty jsou popsány níže.

První komponentou je **predikce incidentu**, která slouží pro zadávání hodnot nezávislých proměnných pomocí formuláře a následného odeslání na backend, jenž pro zadané nezávislé proměnné klasifikuje pomocí naučených modelů (LDA, QDA, RF a NN) do jedné z kategorií závislé proměnné. Výsledky pošle zpět klientovi, který je vypíše. Součástí je také vyhledání

záznamů se stejnými hodnotami (pokud takové existují) a následné vypsání do tabulky spolu se skutečnou hodnotou proměnné „Spent time“. To slouží pro rychlý přehled, kolik úsilí se vynaložilo pro podobný incident. Ukázka komponenty v prohlížeči je na obrázku 21.

Obrázek 21: Komponenta predikce incidentu



Zdroj: Vlastní zpracování, 2023

Druhou komponentou je **predikce úsilí** podle vybraného zákazníka. Komponenta slouží pro navolení anonymizovaného zákazníka z rozbalovacího seznamu, který i udává četnost záznamů k danému zákazníkovi v data setu. Po výběru zákazníka se opět provolá backend aplikace, kde jsou vybrány všechny záznamy, které se řešily u vybraného zákazníka. Kromě výběru také proběhne klasifikace záznamů pomocí již zmíněných naučených modelů a vypočte se průměrné úsilí podle vzorce 5.

$$\overline{st} = \left(\sum_{i=1}^{n} \frac{c_i}{2}\right) \times 8 \tag{5}$$

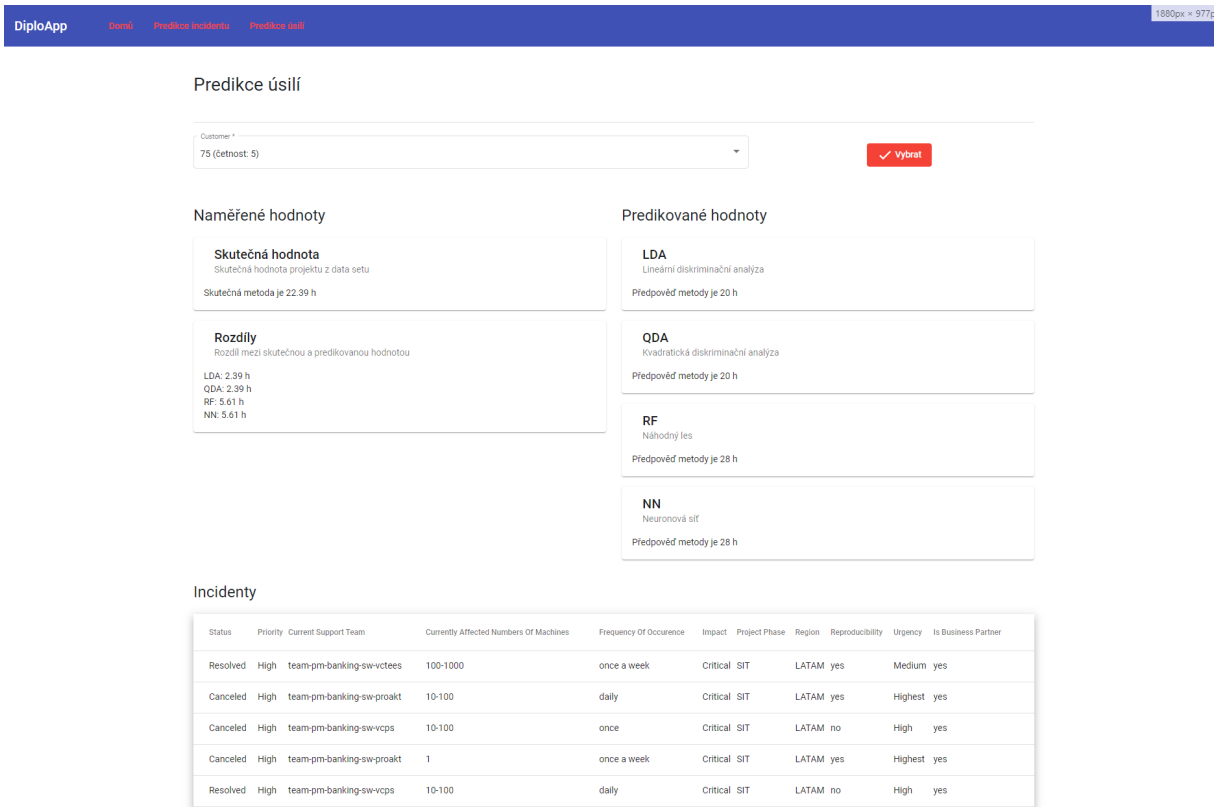
Zdroj: Vlastní zpracování, 2023

kde: n ... počet klasifikačních metod [ks],
$\overline{st}$ ... průměrné úsilí predikované všemi metodami [h],
c_i ... číselná hodnota predikované kategorie i-tou metodou [numerická kategorie].

Navržené metody vracejí číselné hodnoty 1 až 3 podle toho, do jakého intervalu metoda závislou proměnnou klasifikovala. Jak bylo zmíněno v kapitole Změna modelu, hodnota 1 znamená „až“ 1 MD. Incident tedy podle klasifikované kategorie spadá do intervalu 0 až 8 hodin (1 MD), proto je nutné atribut c_i vydělit dvěma, abychom získali průměrnou hodnotu. Jelikož skutečná hodnota „Spent time“ je v hodinách, je nutné ještě výsledek vynásobit 8 (rozpětí intervalu). Takto se získá průměrná predikovaná hodnota.

Zpět na klienta se vrátí kromě průměrné predikované hodnoty také suma skutečných hodnot proměnné „Spent time“, která slouží pro porovnání. Společně se vrátí i hodnoty všech nezávislých proměnných, které se pro informaci vypíší do tabulky. Tímto způsobem lze získat predikci úsilí pro všechny zákazníky a lze je porovnat se skutečnými hodnotami. Ukázka komponenty v prohlížeči je na obrázku 22.

Obrázek 22: Komponenta predikce úsilí



Zdroj: Vlastní zpracování, 2023

10.2.2 Možný budoucí vývoj

Do budoucna je možné aplikaci rozšířit o přímé napojení na systém JIRA pomocí knihovny *Atlasian Python API*, která by umožnila čtení dat přímo z interní databáze systému. Dále je možné také napojit databázi, do niž by se ukládala různá data aplikace. Pomocí protokolu LDAP by bylo také možné využívat správu uživatelských účtů v korporátních sítích, což by umožnilo přihlašování pod jedním firemním účtem. Pro vylepšení klasifikace aplikace by autor

doporučoval rozšířit testovací data, přeučit modely na těchto datech, nebo udělat přímé napojení na JIRU.

10.2.3 Umístění aplikace

Pro vývoj a správu zdrojového kódu byl využit verzovací systém GIT, dostupný přes platformu GitHub. Ten kromě synchronizace kódu (kód je uložený na serveru) a zabezpečení dat nabízí také sledování vývoje a změn v čase. Hlavní předností GITu je také možnost větvení kódu, což umožňuje vytváření větví (anglicky *Branch*) programu, které jsou nezávislé na sobě a slouží pro iterativní vývoj aplikace. Jeden repozitář také umožňuje spolupráci celého týmu, což se dá využít při možném budoucím vývoji aplikace, avšak už v dnešní době je tento přístup standardem.

Repositář aplikací je umístěn na veřejné adrese `https://github.com/vyletat/diplo-app`. Skládá se ze dvou částí — složka „*/server*“, která obsahuje backend aplikace a složka „*/client*“ obsahující frontend aplikace. Aplikace je pod licencí MIT, což umožňuje využití kódu a jeho šíření bez omezení.

V rámci práce je také napsána technická a uživatelská příručka. Technická příručka, která je v Příloze F, obsahuje popis aplikací, včetně struktury a možností spuštění a nasazení na server. Uživatelská příručka, která je v Příloze G, obsahuje návod jak obsluhovat aplikaci z pozice uživatele, tedy neobsahuje technické věci.

11 Interpretace výsledků

V této kapitole jsou porovnány výsledky klasifikačních metod z minulé kapitoly a vyhodnoceny výzkumné otázky definované v kapitole Metodika. Závěr kapitoly obsahuje diskusi a limity řešení z praktické části práce.

11.1 Faktory ovlivňující strávený čas

První výzkumná otázka byla ověřena v kapitolách Korespondenční analýza a Diskriminační analýza. V první zmíněné kapitole byly kategorie nezávislých proměnných promítány společně s kategoriemi závislé proměnné v dvojrozměrném grafu pomocí korespondenčních vztahů a umožnily identifikovat skupiny kategorií, které jsou podobné nebo odlišné v obou souborech proměnných. Ve druhé zmíněné kapitole byla provedena diskriminační analýza, která umožnila odhalit statisticky významné prediktory, které nejvíce přispívají ke správné klasifikaci do kategorie závislé proměnné. Mezi významné prediktory patří „Priority“, „Currently affected numbers of machines“, „Frequency of ocurence“, „Impact“ a „Is business partner“. Tyto zmíněné proměnné můžeme považovat za hlavní faktory, které ovlivňují celkový čas strávený (úsilí) na incidentu. V tabulce 7 jsou seřazeny všechny testované nezávislé proměnné podle jejich schopnosti diskriminovat (rozlišovat), kterou udává F-statistika. Pokud je F-statistika vysoká, znamená to, že rozdíly mezi středními hodnotami mezi skupinami jsou velké v porovnání s variabilitou uvnitř skupin. To naznačuje, že rozdíly mezi skupinami jsou pravděpodobně skutečné a že skupiny jsou statisticky významně odlišné. Naopak, pokud je F-statistika nízká, znamená to, že rozdíly mezi skupinami jsou malé v porovnání s variabilitou uvnitř skupin a rozdíly mezi skupinami nejsou statisticky významné. Pokud je nezávislá proměnná v tabulce označena jako statisticky významná, jedná se o faktor, který ovlivňuje závislou proměnnou ”Spent time”.

Tabulka 7: Výsledky diskriminace proměnných

Pořadí	Proměnná	F-statistika	p-hodnota	Statistická významnost
1.	Impact	7,0631	0,0009	Ano
2.	Frequency of occurence	6,5977	0,0014	Ano
3.	Priority	5,4586	0,0042	Ano
4.	Is business partner	4,9719	0,007	Ano
5.	Currently af. num. of mach.	3,8774	0,0208	Ano
6.	Status	1,6857	0,1854	Ne
7.	Region	1,6461	0,1930	Ne
8.	Current support team	1,4408	0,2369	Ne
9.	Urgency	1,1618	0,3131	Ne
10.	Project phase	0,6486	0,5229	Ne
11.	Reproducibility	0,5359	0,5852	Ne

Zdroj: Vlastní zpracování, 2023

11.2 Klasifikace stráveného času

Pro klasifikaci proměnných byly využity celkem čtyři metody — lineární diskriminační analýza, kvadratická diskriminační analýza, náhodný les a neuronová síť. Klasifikace probíhala ve stejném poměru trénovacích a testovacích dat. Hlavní metrikou pro porovnání úspěšnosti klasifikací je přesnost (popsána v kapitole Diskriminační analýza). Dalšími metrikami jsou průměrná přesnost (anglicky *Macro Average Precision*) a průměrná úplnost (anglicky *Macro Average Recall*). Průměrná přesnost značí podíl počtu správně klasifikovaných pozorování k celkovému počtu pozorování pro každou třídu zvlášť a následné zprůměrování těchto hodnot pro všechny třídy. Stejně tak průměrná úplnost vyjadřuje poměr správně klasifikovaných pozitivních případů ke všem skutečným pozitivním případům v testovacích datech. Tyto metriky nám umožňují hodnotit celkový výkon klasifikátoru bez ohledu na nevyváženost tříd v datasetu. Poslední metrikou je F-skóre (anglicky *F score*), což je harmonický průměr přesnosti a úplnosti zmíněný již v kapitole Náhodný les. Vyjadřuje celkovou kvalitu klasifikace. Pro výpočet těchto metrik je potřeba vytvořit matice chyb (anglicky *Error Matrix*). Matice chyb (tabulka 8) obsahuje počty správně a špatně klasifikovaných příkladů v každé z kategorií.

Tabulka 8: Matice chyb jednotlivých metod

(a) LDA

	1	2	3	
1	**750**	0	0	750
2	230	**1**	0	231
3	26	0	**0**	26
	1006	1	0	1007

(b) QDA

	1	2	3	
1	**736**	7	7	750
2	220	**5**	6	231
3	24	1	**1**	26
	980	13	14	1007

(c) RF

	1	2	3	
1	**673**	67	10	750
2	193	**34**	4	231
3	21	4	**1**	26
	887	105	15	1007

(d) NN

	1	2	3	
1	**707**	43	0	750
2	214	**17**	0	231
3	24	2	**0**	26
	945	62	0	1007

Zdroj: Vlastní zpracování, 2023

Na ose *x* jsou skutečné hodnoty, na ose *y* jsou pak predikované hodnoty, v chybové matici bylo celkem 1 007 hodnot. Na diagonále jsou pak hodnoty, které byly modelem predikovány správně. Všechny hodnoty ležící mimo diagonálu nezvládl daný model klasifikovat správně. Tabulka 9 ukazuje nejlepší naměřené metriky jednotlivých metod.

Z tabulky 9 podle naměřených metrik je vidět, že nejlépe klasifikuje metoda LDA s přesností 74,578 %. Na druhém místě je metoda QDA, třetí je NN a na posledním místě pak RF. Avšak pokud bychom hodnotili kvalitu metod pouze podle metriky přesnosti (*Accuracy*), vynechali

Tabulka 9: Metriky navržených metod

Metoda	Přesnost (*Accuracy*) [%]	Průměrná přesnost (*Precision*)	Průměrná úplnost	Průměrné F-skóre
LDA	74,578	0,5818	0,3348	0,2876
QDA	73,6842	0,4024	0,3475	0,314
RF	70,3078	0,3831	0,361	0,3578
NN	71,8967	0,3408	0,3388	0,3168

Zdroj: Vlastní zpracování, 2023

bychom fakt, že sice metoda RF skončila jako nejhůře klasifikující metodou, ale podle vypočítaného F-skóre vyšla jako nejlépe klasifikující. Toto je dané tím, že v případě nevyvážených tříd je F-skóre lepším ukazatelem. Z matice chyb je také vidět, že jako jediná byla schopna zařadit do druhé kategorie nejvíce záznamů správně. Dále je u ní vidět, že se snažila klasifikovat více záznamů do třetí kategorie než ostatní modely.

11.3 Predikce stráveného času u zákazníků

Výzkumná otázka byla ověřena pomocí vytvořené aplikace. Díky metodám napsaných pro modul „predikce úsilí“ lze pro vybraného zákazníka predikovat celkový čas strávený na řešení incidentů a porovnat ho se skutečně vynaloženým časem.

Pro ověření výzkumné otázky byly vybrány pouze zákazníci, kteří v poskytnutém datasetu mají dva a více incidentů. Z celkového počtu 998 zákazníku se jedná o 374. V opačném případě by se jednalo pouze o predikci jednoho incidentu.

Pro porovnání, jestli predikované hodnoty odpovídají skutečným hodnotám, se použije t-test. Po provedení testu normality na datech se zjistilo, že normalita u proměnných je porušena, tudíž se vybere neparametrický Mann-Whitneyův U-test (M-W). Test porovnává mediány dvou skupin a testuje, zda jsou tyto mediány statisticky významně odlišné. (Svoboda a kol., 2019)

Formuluje se nulová a alternativní hypotéza:

H_0: Medián predikovaného úsilí podle metody *[7] je roven mediánu skutečného úsilí.
H_1: Medián predikovaného úsilí podle metody * není roven mediánu skutečného úsilí.

Z výsledků, které jsou zobrazeny v grafu 10, je vidět, že na základě p-hodnoty nezamítáme nulovou hypotézu pouze pro metodu **RF**. Pro ostatní metody zamítneme nulovou hypotézu na úrovni α. O metodě RF můžeme tedy prohlásit, že průměr predikovaných metod se neliší od hodnot skutečných, a jako o jediné lze prohlásit, že podle ní se dá predikovat úsilí (náklady) na celý projekt podle zákazníka. Na druhém místě se umístila metoda NN, následována QDA

[7] Symbol * znamená nahrazení názvem z jedné metod - LDA, QDA, RF nebo NN.

Tabulka 10: Výsledky M-W testu

Metoda	Medián skup. 1	Medián skup. 2	Statistika U	p-hodnota
LDA	7,2899	4	444614	0,000025
QDA	7,2899	4	447726	0,000074
RF	7,2899	4	479805	0,153533
NN	7,2899	4	457960	0,001625

a na posledním místě skončila metoda LDA podle hodnoty U, která udává jak moc jsou si dvě srovnávané skupiny podobné nebo odlišné.

V rámci druhé a třetí výzkumné otázky může autor prohlásit, že nejlépe klasifikující metodou je RF, jelikož podle matice chyb nejlépe klasifikuje data a také jako jediná dokáže predikovat náklady na projekt u zákazníka.

11.4 Diskuse a limity řešení

Faktory ovlivňující strávený čas byly popsány pro LDA, avšak každá z definovaných metod v této práci může mít jiné důležitosti nezávislých proměnných, které ovlivňují klasifikaci závislé proměnné. V RF se důležitost proměnných obvykle měří na základě toho, jak často je proměnná použita pro dělení datového souboru při tvorbě stromů a jak moc zlepšuje čistotu (*gini index*) uzlů v těchto stromech. U NN se důležitost proměnných často odhaduje na základě gradientů (složek derivací) výstupů sítě vzhledem k hodnotám proměnných. Proměnné, které mají větší vliv na změnu výstupů sítě, budou mít vyšší hodnotu důležitosti. Výsledky diskriminace nezávislých proměnných LDA byly projednány s Ing. Davidem Kriváneckem (osobní komunikace 10. 4. 2023), který s výsledky souhlasil a o odlišnosti důležitosti proměnných v ostatních modelech byl informován.

Za výslednou kvalitu klasifikačních modelů nejvíce ovlivňuje typ proměnných vstupujících do modelů. Všechny proměnné jsou nominální, které poskytují méně informací než proměnné kardinální, které mohou nabývat libovolné hodnoty v určitém rozsahu. To umožňuje kardinálním proměnným poskytnout detailnější popis dat a umožňuje klasifikátoru rozlišovat mezi jednotlivými hodnotami.

Další prostor ke zlepšení klasifikace modelů jsou data jako taková. Největší problém v této oblasti autor spatřuje v rozdělení kategorií závislé proměnné, kde většina dostupných záznamů patří do první třídy, to způsobuje, že model klasifikuje častější třídy s větší přesností a méně časté třídy s menší přesností, což ukazují i výsledky klasifikací zobrazené v matici chyb. Jedním z řešení, by bylo mít k dispozici větší počet dat (např. za 5 let) a rovnoměrně rozdělit kategorie v závislé proměnné a následně přeučit modely.

Modely byly testovány na stejné množině trénovacích a testovacích dat. Jak již autor zmiňoval, byly rozděleny 7:3 a pro 10 000 interací byla vybrána 7 103. iterace parametru (čísla), který náhodně zamíchává data před rozdělením do množin, protože na ní vycházely výsledky modelů nejkvalitněji. Nebyla prováděna žádná validace modelů, která se může provádět např. křížovou validací nebo bootstrapingem. Obecně je cílem validace získat objektivní odhad výkonnosti modelu na nezávislých datech a minimalizovat riziko přetrénování.

Konfigurace může být složitá, jelikož existuje nekonečné množství různých konfigurací a architektur neuronových sítí, což znamená, že nalezení optimálního modelu může být náročné a často vyžaduje několik pokusů a chyb. To je dáno tím, že neurony v síti jsou propojeny a váhovány různými způsoby, což umožňuje sítím zpracovávat složité vzorce a abstraktní koncepty. Kromě toho mohou mít neuronové sítě různé typy vrstev. To dává velkou flexibilitu při vytváření modelů, ale také může být zdrojem komplikací a nepřesností. V oblasti neuronových sítí a jejich konfigurace existuje stále mnoho prostoru pro další výzkum a objevování nových technik a postupů. Například je možné zkoumat nové typy vrstev a architektur, které by mohly být účinnější při zpracování určitých typů dat.

Současný model predikce nákladů by se dal upravit s použitím navržených modelů tak, že by se použila místo počtu ticketů a průměrné hodnoty úsilí hodnota predikovaná vybranou metodou na základě historických dat z podobného projektu s možnou úpravou hodnot proměnných (např. změnil by se současný support tým, projektová fáze atd.). Avšak použití této metody a implementace (pokud se ji rozhodne nějak implementovat do svého současného modelu) je už jen v kompetenci DN.

Závěr

Náklady spojené s údržbou (anglicky *Maintenance*) a podporou (anglicky *Support*) jsou významným faktorem ovlivňujícím celkové náklady provozu a údržby systémů či zařízení. Správné řízení a optimalizace procesů v této oblasti může vést ke snížení nákladů a zvýšení efektivity provozu. Analýza dat je klíčovým nástrojem pro identifikaci trendů, slabých míst a možnosti zlepšení se v této oblasti. Trendem je posun od reaktivního modelu údržby k prediktivnímu modelu, který využívá technologie, jako je umělá inteligence a strojové učení, k predikci poruch nebo potřeby údržby na základě analyzovaných dat.

Celý proces údržby a podpory je součástí životního cyklu softwaru (SDLC). Po nasazení softwaru do produkčního prostředí začíná fáze podpory, která zahrnuje sledování provozu, řešení chyb, opravy, aktualizace, monitorování výkonu a zajištění dostupnosti. Efektivita této fáze závisí na kvalitě a stabilitě softwaru, který byl vyvíjen v předchozích fázích SDLC. Pokud software nebyl dobře navržený, vytvořený a otestovaný, bude na oddělení údržby přicházet více incidentů od zákazníků a bude nutná oprava těchto chyb a tím vzniknou přebytečné náklady, kterým se dalo zabránit. Spokojenost zákazníka s úrovní podpory je klíčovým faktorem pro udržení dobrého vztahu a dlouhodobého partnerství. Oddělení podpory musí tedy rychle a efektivně reagovat na incidenty, které zákazníci nahlásí, a přijímat opatření k jejich vyřešení.

Vzhledem k omezené dostupnosti české literatury na dané téma byl autor nucen při psaní této práce čerpat zejména ze zahraniční literatury. Kvůli aktuálnosti informací byly také využity webové zdroje a články. K popisu rámce ITIL byla použita literatura vydavatele *Axelos*. Pro charakteristiku společnosti byly použity konzultace s Ing. Davidem Kriváknou, interní materiály a částečně i dříve napsaná bakalářská práce autora.

Tato diplomová práce navazuje na bakalářskou práci autora, jež se zabývala rámcem ITIL, který je zde popsaný v nejnovější čtvrté verzi, která se více zaměřuje na metody Lean, Agile a DevOps. V bakalářské práci bylo zmíněné pokračování pomocí prediktivních klasifikačních modelů, přesněji metodami strojového učení, avšak v oblasti prioritizace zákazníků. Autor těchto metod využil, a to v rámci zkoumání probíhajícího v této diplomové práci.

Práce si za hlavní cíl stanovila provést analýzu nákladů na podporu a údržbu a demonstrovat predikci těchto nákladů pomocí vytvořeného programu. Cíle bylo dosaženo statistickým výzkumem a navržením metod klasifikace pro celkově strávený čas (úsilí) na incidentu, podle kterého se počítají náklady. Také byla vytvořena aplikace, která na základě zadaných dat a s použitím navržených metod zmíněné úsilí predikuje.

Pomocí diskriminační analýzy byly objeveny faktory (nezávislé proměnné), které nejvíce přispívají k diskriminaci tříd závislé proměnné, tudíž tyto faktory nejvíce přispívají ke správné klasifikaci. Pro nalezení nejlépe klasifikující metody byly použity metriky přesnost (anglicky *Accuracy*) a F-skóre. Podle přesnosti vyšla nejlépe klasifikující metoda LDA, následována QDA, NN a RF. Avšak tato metrika nezohledňovala nevyváženost tříd v testovacích datech, a proto byla lépe vypovídající metrikou F-skóre. Podle této metriky se umístila metoda RF na prvním místě, jelikož v porovnání s ostatními metodami klasifikovala nejvíce případů i do ostatních tříd. K odhalení, která z metod nejlépe predikuje celkově strávený čas u zákazníků, byl využit neparametrický t-test. Z výsledků testu se jako jediná nezamítla nulová hypotéza na zvolené hladině α u metody RF, a tudíž lze o ní prohlásit, že jako jedinou ji jde využít pro predikci celkově stráveného času.

Práce se zaměřovala na statistickou analýzu poskytnutých dat a nalezení vhodných metod klasifikace stráveného času. Práce se příliš nezaměřuje na optimalizaci metod strojového učení, proto je zde prostor pro další zpracování. Optimalizace může být provedena na různých úrovních, včetně výběru a transformace příznaků, volby modelu, ladění parametrů, techniky učení a evaluace výsledků.

„*Výdaje na podporu a údržbu nejsou jen náklady, ale také investicí do udržitelnosti a kvality celkového softwarového produktu.*“

— Tomáš Vyleta

Seznam literatury

Andreev, V. (2022). *Software maintenance costs: Factors & ways to reduce*. Maddevs. https://maddevs.io/customer-university/software-maintenance-costs/

April, A., Hayes, J. H., Abran, A., & Dumke, R. (2005). Software Maintenance MaturityModel (SMmm): the software maintenance process model. *Journal of Software Mainte-nance and Evolution: Research and Practice*, *17*(3), 197–223. https://doi.org/10.1002/smr.311

Axelos (2019). *ITIL foundation* (4. vyd.). The Stationery Office.

Blue Partners (2022). *Co Je to sla?*. https://www.bluepartners.cz/slovnik-it-pojmu/sla

Cast Software (2022). *The 4 types of Software Maintenance & How They Help: Cast Software*. https://www.castsoftware.com/glossary/Four-Types-Of-Software-Maintenance-How-They-Help-Your-Organization-Preventive-Perfective-Adaptive-corrective

Dehaghani, S., & Hajrahimi, N. (2013). Which Factors Affect Software Projects Mainte-nance Cost More? *Acta Informatica Medica*, *21*(1), 63. https://doi.org/10.5455/aim.2012.21.63-66

Diebold Nixdorf (2020). *M&S Estimations*. Interní prezentace podniku Diebold Nixdorf.

Diebold Nixdorf (2023a). *About us: Who we are: Diebold nixdorf*. https://www.dieboldnixdorf.com/en-us/about-us/who-we-are/

Diebold Nixdorf (2023b). *JSME Global Center Pilsen: Diebold Nixdorf*. https://www.dnpilsen.cz/o-nas/

Diebold Nixdorf (2023c). *Quality Statement*. Tištěný obrázek.

Elliott, G. (2004). *Global business information technology: an integrated systems approach*. Pearson Addison Wesley.

Encyclopedia (2021). *Životní cyklus Vývoje systémů*. Dostupné 6. 12. 2022 z https://wikijii.com/wiki/Systems_development_life_cycle

Encyclopedia (2022). *DevOps*. Dostupné 7. 12. 2022 z https://wikijii.com/wiki/DevOps

Gangur, M., & Svoboda, M. (2021). *Diskriminační analýza*. Výukový materiál k předmětu KEM/VAED.

Grubb, P., & Takang, A. A. (2007). *Software maintenance: concepts and practice* (2. vyd.). World Scientific.

Gunja, S. (2022). *What is devops? unpacking the purpose and importance of an IT cultural revolution*. Dynatrace news. https://www.dynatrace.com/news/blog/what-is-devops/

Hendl, J. (2015). *Přehled statistických metod* (5. vyd.). Portál.

Holek, T. (2007). *Procesní řízení it služeb*. SystemOnLine. https://www.systemonline.cz/sprava-it/procesni-rizeni-it-sluzeb.htm

Hudec, J., Kufner, V., Volný, I., Tichavská, V. L., Vráželová, L., Sýkorová, T., Šveřepa, J., Sládek, A., Navrátil, P., Lukáč, L., & Jaroch, Š. (2012). *ITIL – výkladový slovník a zkratky v češtině*. itSMF Czech Republic. Dostupné z https://is.muni.cz/el/1433/podzim2015/PV214/um/itil_2011_czech_glossary_v2.0.pdf

Hüttermann, M. (2012). *DevOps for Developers*. Apress.

Irwin, L. (2019). *The evolution of ITIL: How the Framework has reshaped IT service management*. IT Governance Blog En. https://www.itgovernance.eu/blog/en/the-evolution-of-itil-how-the-framework-has-reshaped-it-service-management

Klaschka, J., & Kotrč, E. (2004). *Klasifikační a regresní lesy*. https://www.statspol.cz/robust/robust2004/klaschka.pdf

Komenda, M. (2014). *Korespondenční analýza*. https://is.muni.cz/www/98951/41610771/43823411/43823458/Analyza_a_hodnoc/44563155/00_Korespondencni_analyza_FINAL.pdf

Kovalenko, A., Klouda, K., & Vačata, D. (2022). *BI-VZD přednáška 3*. https://kam.fit.cvut.cz/bi-vzd/lectures/files/BI-VZD-02-cs-slides.pdf

Kriványka, D. (2018). *Tvorba cenových nabídek*. Interní prezentace podniku Diebold Nixdorf.

Kurzy.cz (2023). *Diebold Nixdorf s.r.o.*. Dostupné 15. 10. 2022 z https://rejstrik-firem.kurzy.cz/25784528/diebold-nixdorf-sro/

Kushnir, A. (2021). *Software maintenance cost: What is it and why is it so important?*. Bamboo agile. https://bambooagile.eu/insights/software-maintenance-costs/

LiveAgent (2022a). *Service level agreement (SLA)*. https://www.live-agent.cz/funkce/service-level-agreement-sla/

LiveAgent (2022b). *Softwarová Podpora*. https://www.live-agent.cz/slovnik-pojmu-zakaznicke-podpory/softwarova-podpora/

Management Mania (2015). *Produkt*. Dostupné 17. 10. 2022 z https://managementmania.com/cs/produkt

Management Mania (2016). *Služba (service)*. Dostupné 17. 10. 2022 z https://management mania.com/cs/sluzba

Management Mania (2018). *Člověkoden (man-day)*. Dostupné 26. 2. 2023 z https://manage mentmania.com/cs/clovekoden-manday

Martin, M. (2023). *Software development life cycle (SDLC) phases & models*. Guru99. https: //www.guru99.com/software-development-life-cycle-tutorial.html

McKinney, W. (2018). *Python for Data Analysis* (2. vyd.). O'Reilly Media.

Merrill, C. (2022). *Software maintenance: Perfective, adaptive, Corrective & Preventive*. Zibtek. https://www.zibtek.com/blog/software-maintenance-understanding-the-4-m ain-types/

Nagar, T. (2022a). *How to analyze software maintenance costs in 2022?*. Dev Technosys. http s://devtechnosys.com/insights/software-maintenance-costs/

Nagar, T. (2022b). *How much does it cost to hire dedicated software developer in 2022?*. Dev Technosys. https://devtechnosys.com/insights/cost-to-hire-a-software-devel oper/

Niessink, F., & van Vliet, H. (2000). Software maintenance from a service perspective. *Journal of Software Maintenance: Research and Practice*, *12*(2), 103–120. https://doi.org/10.1 002/(SICI)1096-908X(200003/04)12:2<103::AID-SMR205>3.0.CO;2-S

Omeyer, A. (2021). *Software maintenance types: Corrective, adaptive, perfective, and preventive*. Hackernoon. https://hackernoon.com/what-do-you-need-to-know-about-softw are-maintenance-types-as-an-engineer-421335f1

Poláčková, J., & Jindrová, A. (2010). Vyhodnocení dotazníkového šetření pomocí korespondenční analýzy. *Ekonomická revue*, *13*(3), 173-178. https://doi.org/10.7327/cerei.20 10.09.06

Preston, M. (2021). 7 *phases of the System Development Life Cycle Guide*. CloudDefense AI. https://www.clouddefense.ai/blog/system-development-life-cycle

Procházka, J., & Klymeš, C. (2011). *Provozujte IT jinak*. Grada.

Sajna, K. (2022). *The 7 stages of the software development life cycle*. Codilime. https://co dilime.com/blog/the-stages-of-the-sdlc/

Singlemind (2022). *Support & Maintenance*. https://www.singlemindconsulting.com/s ervices/support-and-maintenance/

Software Testing Help (2023). *What is SDLC (software development life cycle) Phases & Process*. https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/

Spinnaker Support (2019). *Defining software support and managed services - what's the difference?*. https://uk.spinnakersupport.com/blog/2019/08/08/what-is-the-difference-between-software-support-and-managed-services/

StatSoft (2013). *Úvod do neuronových sítí*. http://www.statsoft.cz/file1/PDF/newsletter/2013_02_05_StatSoft_Neuronove_site_linky.pdf

Svoboda, M., Gangur, M., & Mičudová, K. (2019). *Statistické zpracování dat*. Západočeská univerzita v Plzni.

Takyar, A. (2022). *How to reduce software maintenance cost of existing software?*. LeewayHertz. https://www.leewayhertz.com/reduce-software-maintenance-cost/

Tayllorcox (2022). *Co Je to itil? Vše O Metodice a ITIL 4 update*. https://www.tx.cz/itil/metodika

Thales Group (2022). *4 Types of Software Maintenance & What is Software Maintenance?*. https://cpl.thalesgroup.com/software-monetization/four-types-of-software-maintenance

Thales Group (2023). *What is a software maintenance process? 4 types of software maintenance*. https://cpl.thalesgroup.com/software-monetization/four-types-of-software-maintenance

TIBCO Software (2023). *What is a random forest?*. Dostupné 20. 11. 2022 z https://www.tibco.com/reference-center/what-is-a-random-forest

Tutorials Point (2022). *Software maintenance overview*. https://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm

Vyleta, T. (2020). *Prioritizace zákazníků při poskytování SW podpory* [Bakalářská práce, Západočeská univerzita v Plzni]. Digitální knihovna Západočeské univerzity v Plzni. https://otik.zcu.cz/handle/11025/40849

Wikipedie (2023a). *Diebold Nixdorf*. Dostupné 15. 10. 2022 z https://en.wikipedia.org/wiki/Diebold_Nixdorf

Wikipedie (2023b). *Nixdorf Computer*. Dostupné 15. 10. 2022 z https://en.wikipedia.org/wiki/Nixdorf_Computer

Wikipedie (2023c). *Software*. Dostupné 17. 10. 2022 z https://cs.wikipedia.org/wiki/Software

Wikipedie (2023d). *Umělá neuronová síť*. Dostupné 28. 2. 2023 z https://cs.wikipedia.org/wiki/Uml_neuronov_s

Seznam tabulek

Seznam obrázků

Seznam použitých zkratek

APAC — *Asia-Pacific*, Asijský Pacifik

API — *Application Programming Interface*, rozhraní pro programování aplikací

ATM — *Automated Teller Machine*, bankomat

CART — *Classification and Regression Trees*, klasifikační a regresní stromy

CCTA — *Central Computing and Telecommunications Agency*

CCTV — *Closed-Circuit Television*, kamerový systém

CM — *Customizing Maintenance*, projektová údržba

COBIT — *Control Objectives for Information and related Technology*

CSS — *Cascading Style Sheets*

CSV — *Comma Separated Value*

DN — Diebold Nixdoft

EaFAT — E*ffort and Financial Assessment Tool*

EBS — *External Business Support*

EDA — *Exploratory Data Analysis*, explorační analýza dat

EMEA — *Europa, Middle East, Africa*, Evropa, Střední východ a Afrika

FAQ — *First Qualified Answer*, první kvalifikovaná odpověď

FTP — *File Transfer Protocol*

GC — *Global Center*

GCCC — *Global Custemer Call Center*

GES — *Global Enginering Support*

HTML — *Hypertext Markup Language*

HTTP — *Hypertext Transfer Protocol*

IQR — *Interquartile Range*, mezikvartilové rozpětí

ISO — *International Organization for Standardization*

IaaS — *Infrastructure as a Service*, integrace jako služba

IT — *Information Technology*, informační technologie

ITIL — *Information Technology Infrastructure Library*

ITSM — *IT Service Management*, správa služeb IT

JQL — *Jira Query Language*

K-S — Kolmogorov-Smirnovův Lillieforsův test

LATAM — *Latin America*, Latinská Amerika

LDA — *Linear Discriminant Analysis*, lineární diskriminační analýza

LDAP — *Lightweight Directory Access Protocol*

M&S — *Maintenance & Support*, údržba a podpora

MD — *Manday*, člověkoden

MIT — *Massachusetts Institute of Technology*

MLP — *Multilayer Perceptron*

MLRA — *Multidimensional Linear Regresion Analysis*, vícerozměrná lineární regresní analýza

M-W — Mann-Whitneyův U-test

NA — *North America*, Severní Amerika

NN — *Neural Network*, neuronová síť

NPM — *Node Package Manager*

NYSE — *New York Stock Exchange*, Newyorská burza cenných papírů

PHP — *Hypertext Preprocessor*

PM — *Product Maintenance*, údržba produktu

QA — *Quality Assurance*, zajištění kvality

QDA — *Quadratic Discriminant Analysis*, kvadratická diskriminační analýza

R&D — *Research & Development*, výzkum a vývoj

ReLU — *Rectified Linear Unit*

RF — *Random forrest*, náhodný les

Sass — *Syntactically Awesome Style Sheets*

SDLC — *Software Development Life Cycle*, životní cyklus vývoje softwaru

SIT — *System Integration Testing*, systémové integrační testy

SLA — *Service Level Agreement*, dohoda o úrovni služeb

SLM — *Service Level Management*, správa úrovně služeb

SRS — *Software Requirements Specification*, specifikace softwarových požadavků

SSH — *Secure Shell*

SVS — *Service Value System*

SW — *Software*, software

UAT — *User Acceptance Tests*, uživatelské akceptační testy

UI — *User Interface*, uživatelské rozhraní

URL — *Uniform Resource Locator*, jednotný lokátor zdroje

USA — *United States of America*, Spojené státy americké

W — Wilcova lambda

Seznam příloh

Příloha A: Ukázka dat

Kvůli velikosti tabulky byla rozdělena na dvě části. Obsah přílohy začíná až na druhé, kvůli orientaci na šířku.

Tabulka 11: Ukázka dat a)

Spent time	Customer	Product	Status	Priority	Current sup. team	Curr. aff. num. of mach.	
5,86	802	133	Resolved	Medium	team-pm-banking-sw-vcps	2-10	...
6,66	363	130	Resolved	Medium	team-pm-banking-sw-vcps	2-10	...
10,11	468	19	Canceled	Low	team-pm-banking-sw-vcps	1	...
0,57	662	134	Resolved	Medium	team-pm-banking-sw-vcps	2-10	...
12,93	76	121	Canceled	Medium	team-pm-banking-sw-vcps	10-100	...
7,87	208	136	Resolved	Low	team-pm-banking-sw-vcps	10-100	...
7,28	373	49	Canceled	Medium	team-pm-banking-sw-vcps	1	...
0,21	901	96	Resolved	Medium	team-pm-banking-sw-vocms	100-1000	...
1,77	267	121	Resolved	Medium	team-pm-banking-sw-vcps	2-10	...
4,82	162	17	Resolved	Medium	team-pm-banking-sw-vcps	100-1000	...
12,45	857	20	Resolved	Medium	team-pm-banking-sw-vcps	2-10	...
6,27	795	138	Resolved	Medium	team-pm-banking-sw-vcps	10-100	...
2,82	571	136	Resolved	Low	team-pm-banking-sw-vcps	1	...
5,57	48	136	Resolved	Medium	team-pm-banking-sw-vcps	2-10	...
...	...	...	...	...	...	...	...
2,94	530	136	Resolved	High	team-pm-banking-sw-vcps	1	...
5,93	922	122	Resolved	Medium	team-pm-banking-sw-vcps	1	...
1,04	871	111	Resolved	Low	team-pm-banking-sw-vcps	2-10	...
1,3	114	40	Resolved	Low	team-pm-banking-sw-vcps	1	...
1,29	23	122	Resolved	Medium	team-pm-banking-sw-vcps	more than 1000	...
0,2	819	165	Resolved	Low	team-pm-banking-sw-vocms	2-10	...
0,2	133	96	Resolved	Medium	team-pm-banking-sw-vocms	100-1000	...
5,39	329	122	Resolved	Medium	team-pm-banking-sw-vcps	1	...
2,2	936	68	Resolved	Medium	team-pm-banking-sw-vcps	100-1000	...
3,49	48	162	Resolved	Medium	team-pm-banking-sw-vss	10-100	...
0,2	981	96	Resolved	Low	team-pm-banking-sw-vocms	1	...
0,2	282	165	Resolved	High	team-pm-banking-sw-vocms	100-1000	...
0,2	314	165	Resolved	Medium	team-pm-banking-sw-vocms	more than 1000	...
0,2	187	51	Resolved	Low	team-pm-banking-sw-vocms	1	...

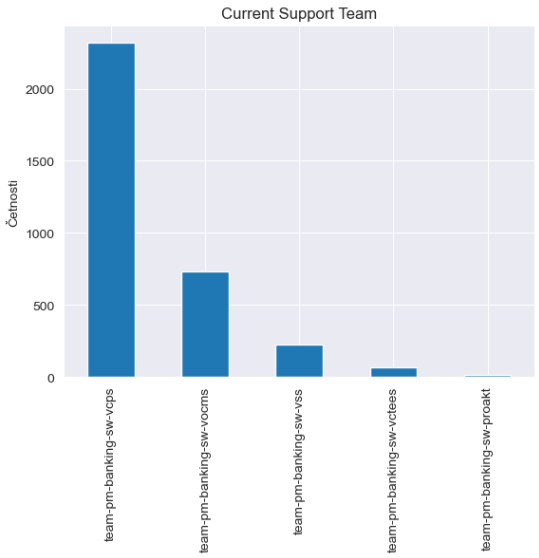
Tabulka 11: Ukázka dat b)

	Frequency of occurence	**Impact**	**Project phase**	**Region**	**Reproducibility**	**Urgency**	**Is bus. par.**
...	always	Non-Critical	Certification	NOA	yes	High	no
...	always	Non-Critical	UAT	APAC	yes	High	no
...	daily	Non-Critical	SIT	EMEA	yes	Medium	yes
...	always	Non-Critical	Pilot	EMEA	yes	High	no
...	daily	Non-Critical	Production	EMEA	yes	High	yes
...	always	Non-Critical	Production	LATAM	yes	Low	no
...	less than once a week	Non-Critical	Production	EMEA	yes	Highest	no
...	once	Non-Critical	Production	EMEA	yes	Low	yes
...	daily	Non-Critical	Internal QA	EMEA	yes	High	yes
...	daily	Non-Critical	Production	EMEA	yes	Medium	yes
...	daily	Non-Critical	Pilot	APAC	no	Highest	yes
...	always	Non-Critical	Production	LATAM	yes	Medium	no
...	always	Non-Critical	Certification	EMEA	yes	Medium	no
...	always	Non-Critical	Pilot	EMEA	yes	Medium	no
...	...	...	...	...	...	...	...
...	always	Non-Critical	Internal QA	APAC	yes	Highest	no
...	once	Non-Critical	Production	APAC	no	Medium	no
...	daily	Non-Critical	Production	APAC	yes	Low	yes
...	once	Non-Critical	Internal QA	EMEA	no	High	yes
...	daily	Non-Critical	Internal QA	APAC	yes	Medium	no
...	always	Non-Critical	Production	APAC	yes	Low	no
...	daily	Non-Critical	Production	EMEA	yes	Medium	yes
...	always	Non-Critical	Pilot	LATAM	yes	Medium	no
...	daily	Non-Critical	Production	APAC	yes	Highest	yes
...	always	Non-Critical	Production	EMEA	yes	Medium	no
...	daily	Non-Critical	UAT	EMEA	yes	Medium	yes
...	always	Non-Critical	Production	EMEA	yes	High	no
...	always	Non-Critical	UAT	APAC	yes	Medium	no
...	once	Non-Critical	UAT	EMEA	yes	Low	yes

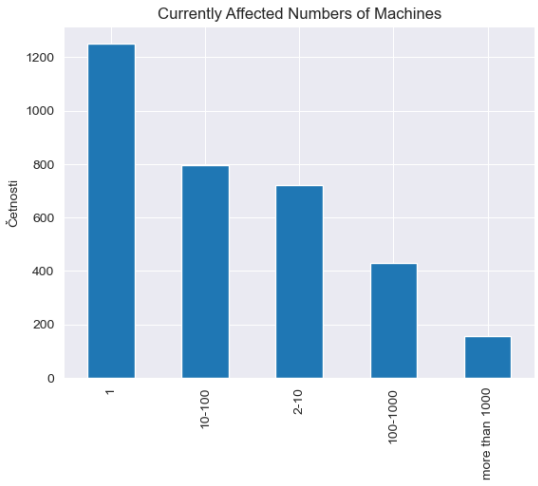
Příloha B: Vizualizace dat

Obrázek 23: Sloupcové grafy a)

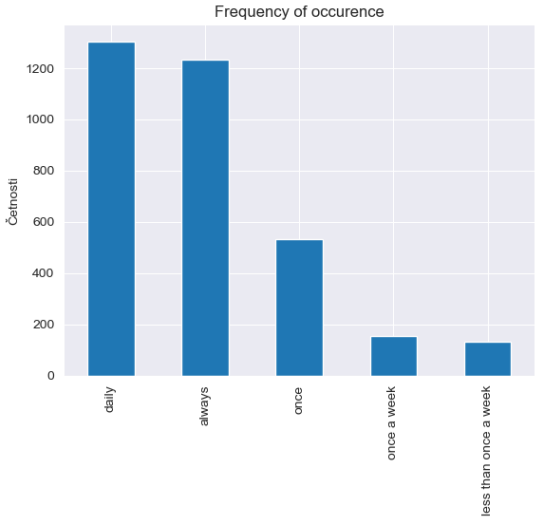
(a) Current Support Team



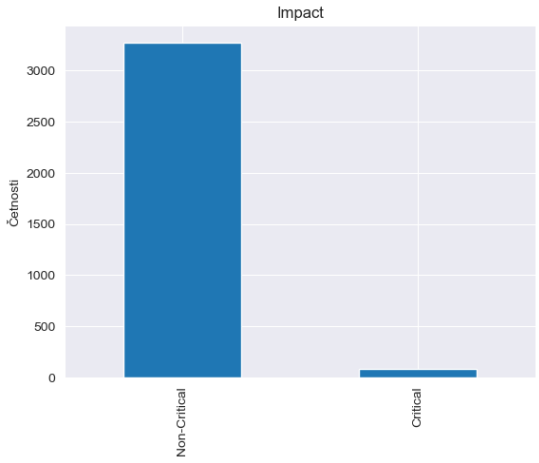
(b) Currently Affected Numbers of Machines



(c) Frequency of occurence

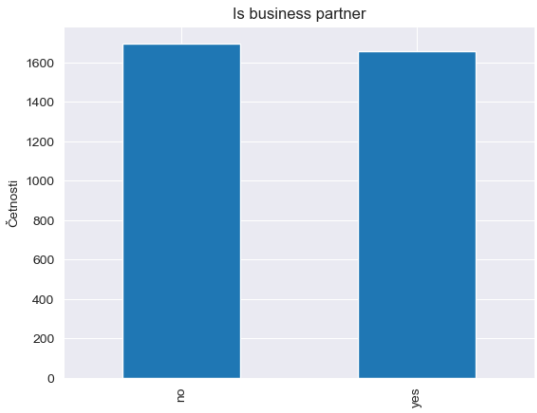


(d) Impact

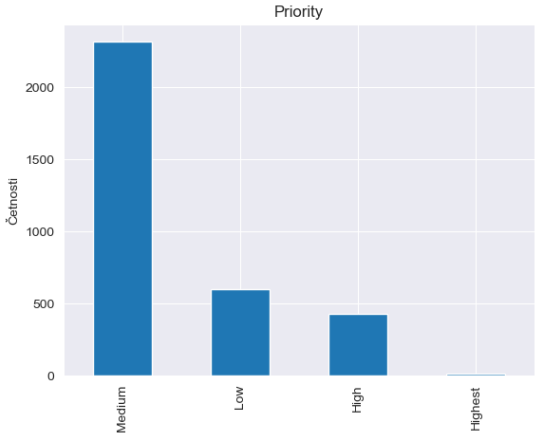


Obrázek 23: Sloupcové grafy b)

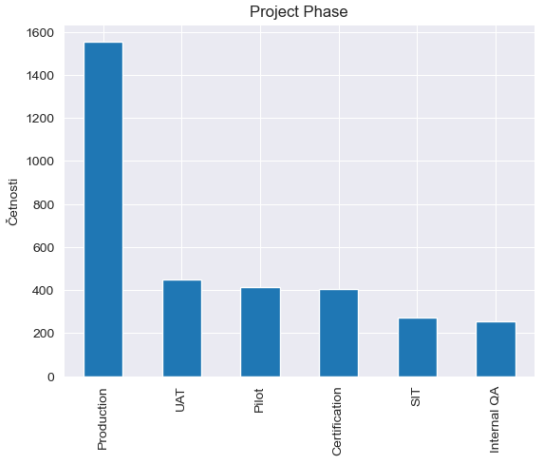
(e) Is Business Partner



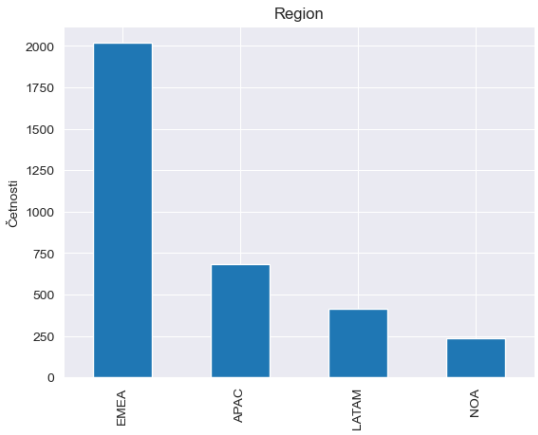
(f) Priority



(g) Project Phase

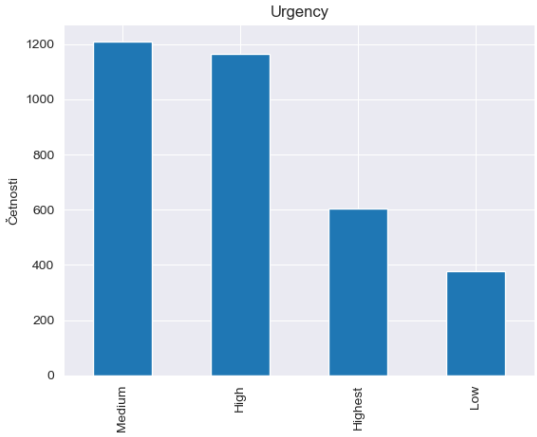


(h) Region

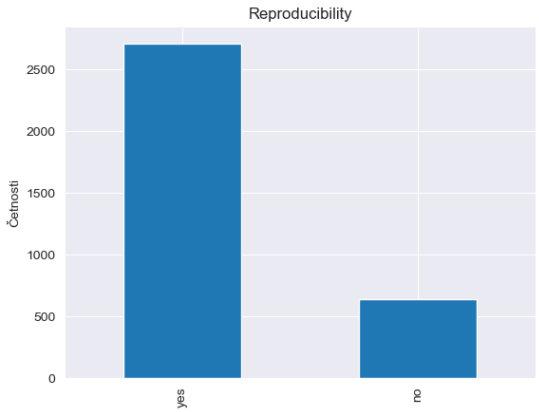


Obrázek 23: Sloupcové grafy c)

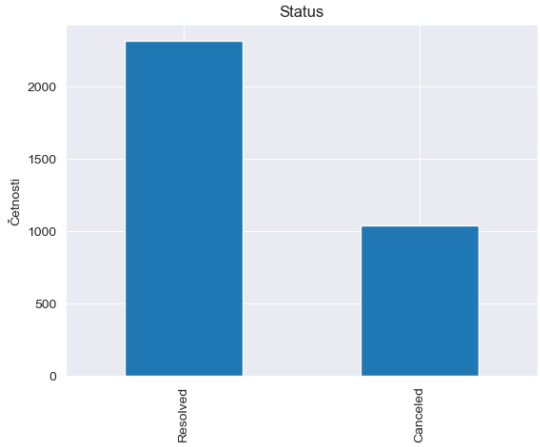
(i) Urgency



(j) Reproducibility



(k) Status



Zdroj: Vlastní zpracování, 2023

Příloha C: Popisná statistika

Tabulka 12: Popisná statistika pro nominální proměnné

Proměnná	Kategorie	Četnost	Procenta
Current Support Team	team-pm-banking-sw-vcps	2320	69,1711
	team-pm-banking-sw-vocms	730	21,7651
	team-pm-banking-sw-vss	226	6,7382
	team-pm-banking-sw-vctees	64	1,9082
	team-pm-banking-sw-proakt	14	0,4174
Currently Affected Numbers of Machines	1	1252	37,3286
	10-100	796	23,7329
	2-10	723	21,5564
	100-1000	429	12,7907
	more than 1000	154	4,5915
Frequency of occurence	daily	1304	38,8790
	always	1231	36,7024
	once	531	15,8318
	once a week	156	4,6512
	less than once a week	132	3,9356
Impact	Non-Critical	3279	97,7639
	Critical	75	2,2361
Is Business Partner	no	1698	50,6261
	yes	1656	49,3739
Priority	Medium	2317	69,0817
	Low	601	17,9189
	High	425	12,6714
	Highest	11	0,3280
Project Phase	Production	1557	46,4222
	UAT	449	13,3870
	Pilot	413	12,3137
	Certification	406	12,1049
	SIT	274	8,1694
	Internal QA	255	7,6029
Region	EMEA	2018	60,1670
	APAC	685	20,4234
	LATAM	415	12,3733
	NOA	236	7,0364
Reproducibility	yes	2714	80,9183

Tabulka 12: Popisná statistika pro nominální proměnné

Proměnná	Kategorie	Četnost	Procenta
	no	640	19,0817
Status	Resolved	2315	69,0221
	Canceled	1039	30,9779
Urgency	High	1164	34,7048
	Medium	1209	36,0465
	Highest	604	18,0083
	Low	377	11,2403

Příloha D: Číselné hodnoty

Tabulka 13: Nahrazení číselnými hodnotami

Proměnná	Textová hodnota	Číselná hodnota
Current Support Team	team-pm-banking-sw-proakt	1
	team-pm-banking-sw-vcps	2
	team-pm-banking-sw-vctees	3
	team-pm-banking-sw-vocms	4
	team-pm-banking-sw-vss	5
Currently Affected Numbers of Machines	1	1
	10-100	2
	100-1000	3
	2-10	4
	more than 1000	5
Frequency of occurence	always	1
	daily	2
	less than once a week	3
	once	4
	once a week	5
Impact	Critical	1
	Non-Critical	2
Is business partner	no	1
	yes	2
Priority	High	1
	Highest	2
	Low	3
	Medium	4
Project Phase	Certification	1
	Internal QA	2
	Pilot	3
	Production	4
	SIT	5
	UAT	6
Region	APAC	1
	EMEA	2
	LATAM	3
	NOA	4

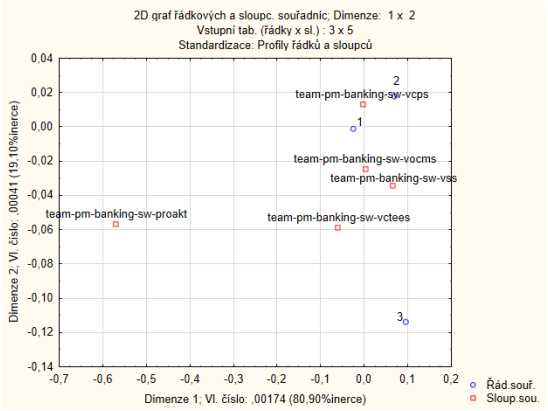
Tabulka 13: Nahrazení číselnými hodnotami

Proměnná	Textová hodnota	Číselná hodnota
Reproducibility	no	1
	yes	2
Status	Canceled	1
	Resolved	2
Urgency	High	1
	Highest	2
	Low	3
	Medium	4

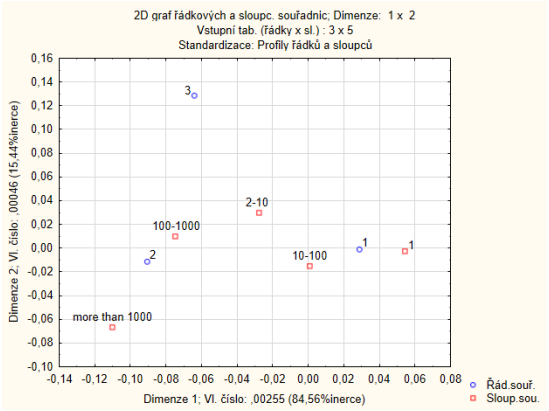
Příloha E: Korespondenční mapy

Obrázek 24: Korespondenční mapy a)

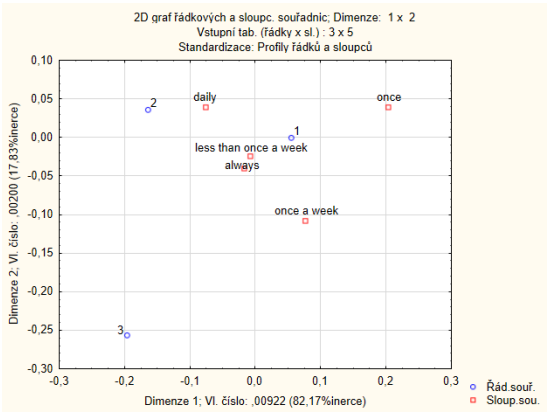
(a) Current Support Team



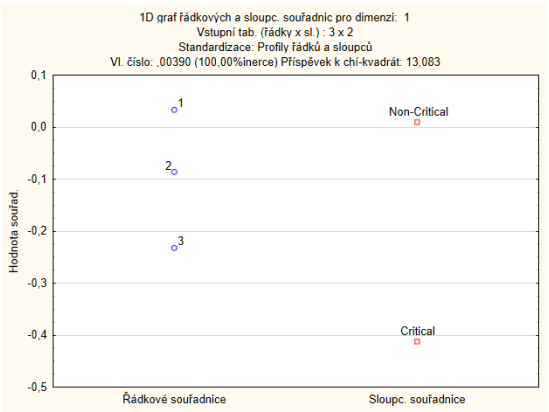
(b) Currently Affected Numbers of Machines



(c) Frequency of occurence

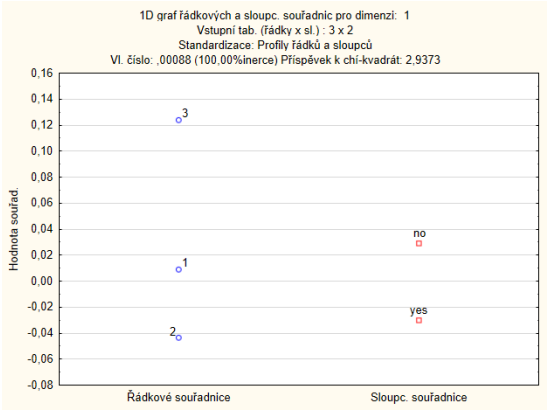


(d) Impact

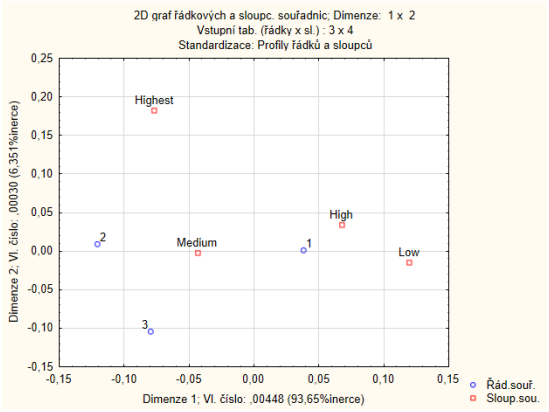


Obrázek 25: Korespondenční mapy b)

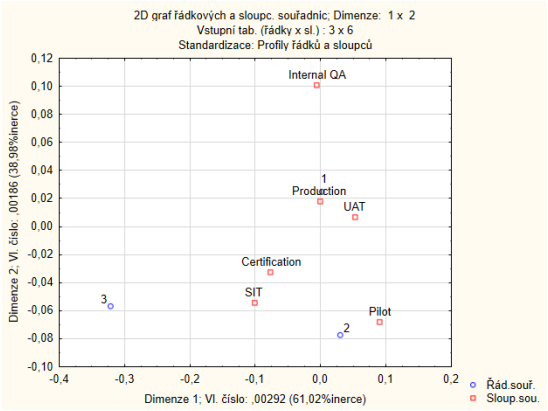
(a) Is Business Partner



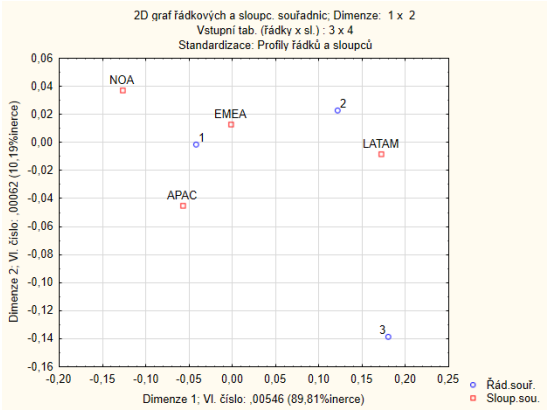
(b) Priority



(c) Project Phase

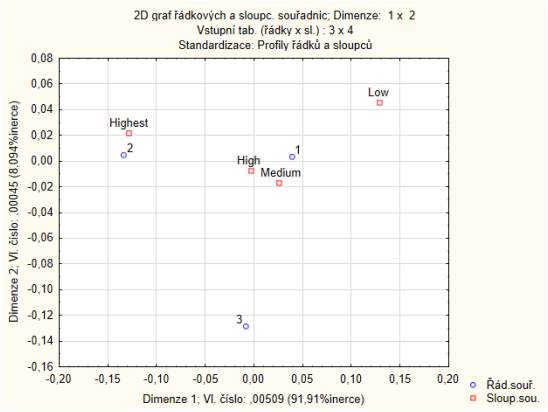


(d) Region

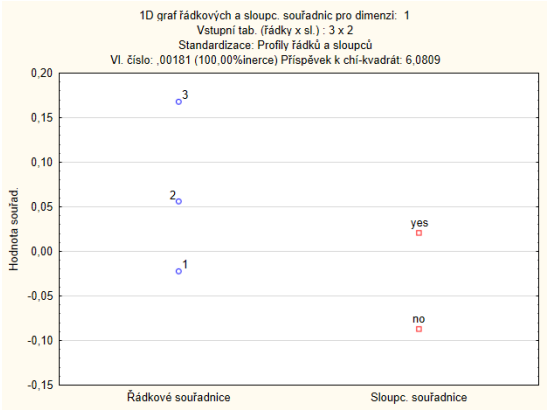


Obrázek 26: Korespondenční mapy c)

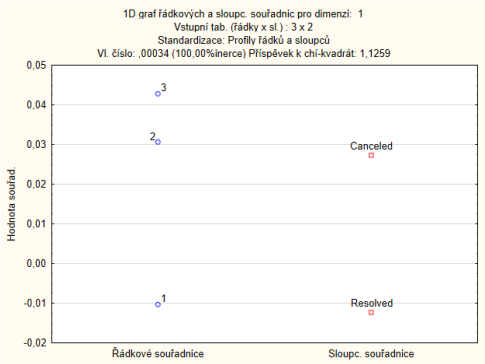
(a) Urgency



(b) Reproducibility



(c) Status



Zdroj: Vlastní zpracování, 2023

Příloha F: Technická příručka

Aplikace s skládá ze dvou částí — frontendové části, která slouží k zadávání dat, a backendové, která slouží pro zpracování a vyhodnocení dat.

Frontend

Frontend je celý napsaný ve frameworku Angular. Angular běží ve verzi **13** a je postavený na Node.js, což je JavaScriptový runtime, který umožňuje běh aplikací na serverové straně. Jazykem pro psaní aplikace je TypeScript, který rozšiřuje syntaxi jazyka JavaScript o silnou typovou kontrolu a další pokročilé funkce, jako jsou například třídy, rozhraní a dekorátory. Nedílnou součástí jsou také moduly, které slouží k organizaci a řízení různých funkcí a služeb v aplikaci a framework je umožňuje sdílet. Moduly mohou být importovány do jiných modulů, což umožňuje aplikaci být rozdělenou na menší logické celky.

Základním kamenem jsou komponenty, které se skládají ze šablony, třídy, CSS souboru a unit testů. Šablona umožňuje definovat vzhled komponenty a také zobrazovat data v HTML. Ve třídě komponenty je uložená logika, instance atd. a také umožňuje napojení na další části aplikace. V CSS soubor jsou definované kaskádové styly použité v šabloně. Unit testy slouží pro testování komponenty.

Další částí jsou také servisy, které umožňují zpracovávat data od externích aplikací, jako je například napojení na backendovou část pomocí HTTP protokolu. Data přicházejí v tzv. *Observable*, který popisuje asynchronní posloupnost hodnot. Je to jako tok dat, který může mít několik hodnot a v průběhu času se může měnit. Další částí jsou také rozhraní, která umožňují strukturovat data, aby nedošlo k chybě.

Spuštění aplikace probíhá tzv. *bootstrapingem*, což je proces inicializace a spuštění aplikace. V této fázi Angular vytvoří instanci aplikace a napojí ji na základní komponentu, která bude sloužit jako kořenový prvek pro celou aplikaci.

Struktura aplikace

Následující graf ukazuje základní strukturu aplikace včetně popisu nejdůležitějších souborů a složek.

```
.
├── src
│   ├── app
│   │   ├── com - složka obsahující komponenty
│   │   │   ├── customer-effort -
│   │   │   │   └── ...
│   │   │   ├── error -
│   │   │   │   └── ...
│   │   │   ├── home -
│   │   │   │   └── ...
│   │   │   └── predict-issue -
│   │   │       └── ...
│   │   ├── inter - složka, ve které jsou definovány rozhraní
│   │   │   └── ...
│   │   ├── ser - složka obsahující servisy
│   │   │   └── ...
│   │   ├── utils - složka obsahující pomocné třídy
│   │   │   └── ...
│   │   ├── app.module.ts - základní modul aplikace
│   │   ├── app.component.css - základní CSS aplikace
│   │   ├── app.component.html - základní šablona aplikace
│   │   ├── app.component.ts - základní třída aplikace
│   │   ├── app-routing.module.ts - modul pro navigaci
│   │   └── ...
│   ├── assets - složka obsahující obrázky atd.
│   ├── index.html - vstupní šablona aplikace
│   ├── proxy.conf.json - nastavení proxy
│   └── ...
├── angular.json - základní nastavení framewroku
├── package.json - nastavení projektu (Node.js)
└── ...
```

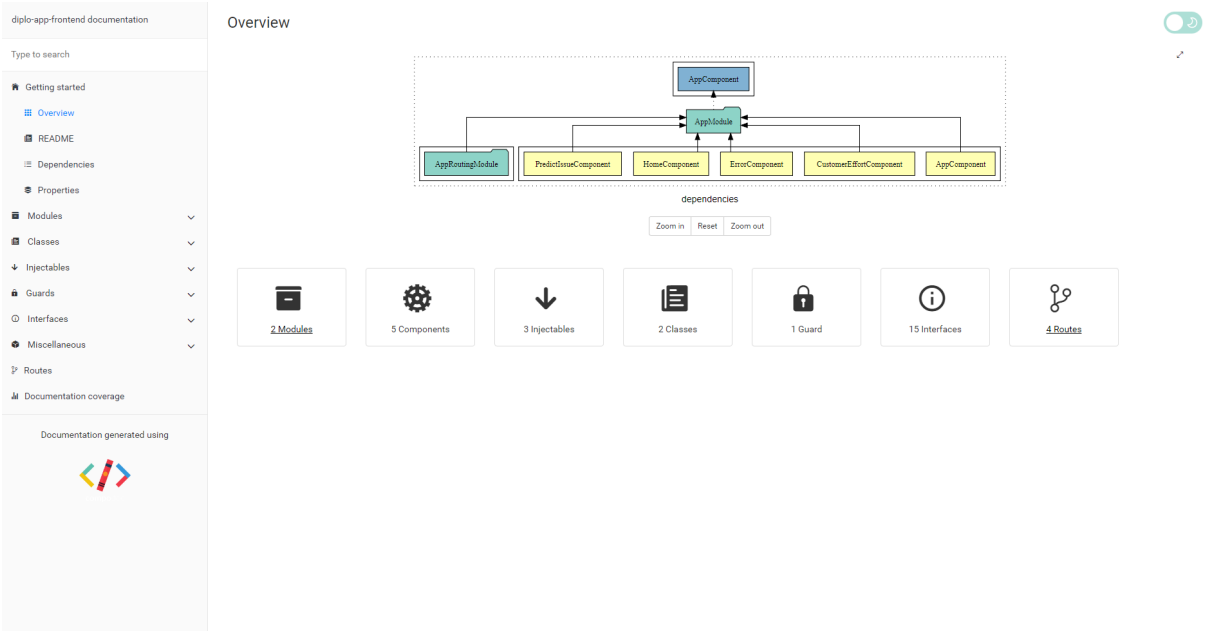
Prerekvizity

Je potřeba mít nainstalovaný zmíněný Node.js nejlépe v nejnovější verzi. V rámci platformy se také stáhne NPM (= *Node Package Manager*), což je správce balíčků pro jazyk JavaScript. Pomocí něho lze snadno instalovat závislosti projektu.

Příkazy

Aplikace se spouští příkazem „*npm run start*“ v kořenové složce, což spustí aplikaci na lokálním prostředí na adrese `http://localhost:4200`[8]. Dalším užitečným příkazem je „*compodoc:build-and-serve*“, který připraví a pustí dokumentaci aplikace, která je přístupná na adrese `http://127.0.0.1:8080`.

Obrázek 27: Dokumentace pro frontend



Zdroj: Vlastní zpracování, 2023

Nasazení

Nasazení aplikace je jednoduché, stačí vytvořit produkční sestavení. Toho lze dosáhnout spuštěním příkazu „*ng build*“, který vytvoří produkční sestavení aplikace v adresáři „*/dist/*“. Tento adresář obsahuje veškerý kód a soubory, které jsou potřebovat pro nasazení aplikace. Pro nasazení na server stačí použít FTP nebo SSH pro nahrání souborů z adresáře na server.

Změna parametrů

Pokud je potřeba, je možné změnit názvy proměnných, jejich textových hodnot i číselných hodnot v souboru „*/src/app/utils/ParameterConfig.ts*“.

[8] Avšak pokud si budete klonovat aplikaci na své zařízení, je nejdříve ještě nutné spustit příkaz „*npm install*“, který nainstaluje všechny závislosti aplikace.

Backend

Backend je napsaný v jazyce Python pomocí frameworku FastAPI pro vytváření API. Pro statistické modely se používají knihovny, jako jsou *Pandas*, *ScyPi*, *Keras*, *Tensorflow* nebo *NumPy*. Backend běží bez propojení na databázi, kde se všechna potřebná data bere z poskytnutého data setu.

Aplikace je rozdělena na třídy spravující API aplikace — poskytují přístupové body pro komunikaci mezi dvěma částmi aplikace. Dále jsou zde servisy pro načtení a inicializaci statistických modelů a třídy pro definování daných modelů. Také jsou zde rozhraní pro správu datových typů příchozích a odchozích dotazů.

Struktura aplikace

```
.
├── app
│   ├── api
│   │   ├── customer_effort_api.py - API třída komponenty predikce úsilí
│   │   ├── predictions_api.py - API třída komponenty predikce incidentu
│   │   └── ...
│   ├── pred_models
│   │   ├── lda_model.py - Model LDA
│   │   ├── qda_model.py - Model QDA
│   │   ├── rf_model.py - Model RF
│   │   ├── nn_model.py - Model NN
│   │   └── ...
│   ├── service -
│   │   ├── data_service.py - Třída pro správu dat
│   │   ├── prepare_service.py - Třída pro inicializaci metod
│   │   └── ...
│   ├── main.py - Hlavní bod aplikace
│   └── schemas.py - Rozhraní
├── data
│   └── input-data.xlsx - Vstupní soubor dat
├── requirements.txt - Závislosti aplikace
└── ...
```

Prerekvizity

Kromě závislostí stačí ke spuštění pouze Python (nejlépe nejnovější verze). V rámci platformy se také stáhne *Pip*, což je správce balíčků pro jazyk Python. Pomocí něho lze snadno instalovat závislosti projektu.

Příkazy

Nejdříve je nutné stáhnout závislosti (stejně jako u frontendu), pro to se využije příkaz „*pip install -r requirements.txt*“ v kořenové složce. Dále je možné spustit hlavní soubor ve složce „*./app*“ pomocí příkazu „*python main.py*“. Na lokálním prostředí aplikace běží na adrese `http://localhost:4444`. Na adrese `http://localhost:4444/docs` také běží API dokumentace, která je automaticky generována.

Obrázek 28: API dokumentace backendu

Zdroj: Vlastní zpracování, 2023

Nasazení

Pokud máte vlastní server, můžete na něj nainstalovat potřebné závislosti (knihovny) pro aplikaci a spustit ji přímo na serveru. Pro správu aplikací napsaných v jazyce Python můžete použít například nástroj *Apache* nebo *nginx*.

Změna parametrů

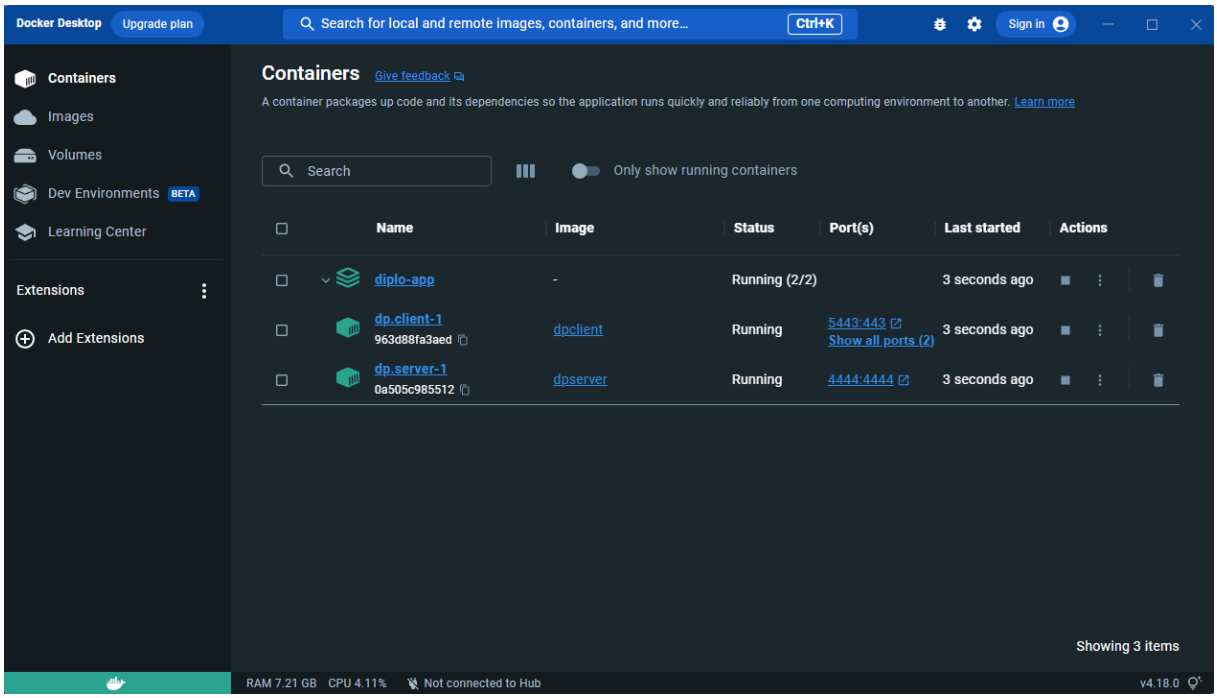
V jednotlivých třídách modelů je možné měnit parametry statistických metod. Vstupní data (data set) je také možné měnit ve složce „*./data*“, avšak je nutné zachovat formát, jinak by se musela předělávat celá aplikace.

Docker nasazení

Docker je technologie pro kontejnerizaci aplikací, která umožňuje spouštět aplikace v izolovaném prostředí zvaném kontejner. Aplikace jsou také společně nasazeny v Dockeru. Tímto způsobem můžeme snadno vytvořit kompletní aplikaci spustitelnou na jedno kliknutí. Při použití Dockeru jsou aplikace spouštěny v izolovaném prostředí, které je stejné na všech počítačích a serverech, což usnadňuje jejich nasazení. Docker také umožňuje jednoduše spravovat závislosti aplikací, protože je můžeme nainstalovat do kontejneru, a tak neovlivní ostatní aplikace na serveru.

Pro spuštění kontejnerů stačí do konzole napsat příkaz „*docker-compose up - -build*“ v kořenové složce repozitáře. Po spuštění je klientská část aplikace dostupná na adrese `http://localhost:5080` a serverová část na adrese `http://localhost:4444`. Ukázka spuštěných kontejnerů je na obrázku 29.

Obrázek 29: Spuštěné kontejnery v Dockeru



Zdroj: Vlastní zpracování, 2023

Příloha G: Uživatelská příručka

V aplikaci existují dvě komponenty, které může uživatel v rámci aplikace využít.

Predikce incidentu

První komponenta se nachází na adrese `http://localhost:4200/predict-issue`, kterou může uživatel napsat do *URL*, nebo k ní přistoupit přes odkaz na domovské stránce.

Zde uživatel vidí formulář, kam se zadávají hodnoty proměnných, všechna políčka jsou povinná, jinak aplikace nedovolí formulář odeslat. Po odeslání se načtou vypočítané hodnoty úsilí pro zadané parametry. Ve spodní části se také může objevit tabulka se stejnými incidenty, pokud existují. Pokud si uživatel přeje hodnoty proměnných změnit, může odeslat nový požadavek a data se přenačtou.

Predikce usilí

Druhá komponenta se nachází na adrese `http://localhost:4200/customer-effort`, kterou může uživatel napsat do *URL*, nebo k ní přistoupit přes odkaz na domovské stránce.

Zde je rozbalovací seznam, který obsahuje všechny zákazníky z data setu včetně jejich četností. Po vybrání zákazníka a odeslání požadavku se zobrazí celkem tři části. První částí jsou průměrné hodnoty predikované metodami na všechny incidenty zákazníka podle vybraného uživatele. V levé části se nachází skutečná hodnota úsilí vynaloženého na incident a rozdíly skutečné a predikované hodnoty pro každou metodu. V dolní části jsou hodnoty proměnných, ze kterých se vycházelo, tedy incidenty vybraného zákazníka.

Příloha H: Obsah DVD

Obsah přiloženého DVD:

- složka, která obsahuje kód webové aplikace a serveru (stejné rozložení jako na GitHubu),
- soubor *input-data.xlsx*, který se používá jako vstup pro backend (je nutné ho dát do složky „*/server/data*“).

Abstrakt

Vyleta, T. (2023). *Analýza nákladů při support a maintenance* [Diplomová práce, Západočeská univerzita v Plzni].

Klíčová slova: analýza nákladů, agilní vývoj, DevOps, Diebold Nixdorf, ITIL, klasifikace, metody strojového učení, podpora, SDLC, software, údržba

Tato diplomová práce se zaměřuje na analýzu nákladů souvisejících s údržbou a podporou softwaru. Téma bylo navrženo společností Diebold Nixdorf, která měla zájem zjistit více informací o složení nákladů na incidenty. Práce obsahuje stručné představení společnosti a představení pojmů souvisejících s vývojem softwaru. Dále je popsán životní cyklus softwaru (SDLC), který zahrnuje údržbu a podporu. V další části je podrobně rozebrána údržba, včetně jejích fází a popisu nákladů. Následuje sekce věnovaná rámci ITIL, který autor již představil ve své bakalářské práci, ale tentokrát se zaměřuje na nejnovější, čtvrtou verzi. V praktické části je představena současná metoda odhadu nákladů společnosti a proveden statistický výzkum na poskytnutých datech. Byly navrženy čtyři metody pro klasifikaci úsilí (celkového času stráveného na incidentu), na základě kterých společnost vypočítává náklady. Součástí práce je také vytvořená aplikace, která na základě zadaných parametrů incidentu klasifikuje toto úsilí a předpovídá celkové úsilí vynaložené na zákazníka. Výsledky výzkumu identifikovaly faktory, které nejvíce ovlivňují klasifikaci úsilí, a jako nejvhodnější metoda pro jeho klasifikaci a odhad celkového úsilí byla vybrána metoda strojového učení nazvaná náhodný les (RF). Společnost projevila zájem o ukázku aplikace a navržených metod. Do budoucna lze na téma diplomové práce navázat vypracováním podrobnější studie metod strojového učení včetně jejich návrhu.

Abstract

Vyleta, T. (2023). *Cost analysis for support and maintenance* [Master's Thesis, University of West Bohemia].

Key words: cost analysis, agile development, DevOps, Diebold Nixdorf, ITIL, classification, machine learning methods, support, SDLC, software, maintenance

This master's thesis focuses on the cost analysis related to software maintenance and support. The topic was suggested by Diebold Nixdorf, which was interested in finding out more about the composition of incident costs. The thesis includes a brief introduction to the company and an introduction to concepts related to software development. Furthermore, the software development life cycle (SDLC), which includes maintenance and support, is described. In the next section, maintenance is discussed in detail, including its phases and cost breakdown. This is followed by a section devoted to the ITIL framework, which the author has already introduced in his bachelor's thesis, but this time focuses on the latest, fourth version. In the practical part, the current method of estimating company costs is presented, and statistical research is performed on the provided data. Four methods have been proposed to classify the effort (total time spent on an incident) on the basis of which the company calculates the cost. The work also includes the development of an application that classifies this effort and predicts the total effort spent per customer based on the given incident parameters. The results of the research identified the factors that most affect the classification of the effort, and a machine learning method called Random Forest (RF) was selected as the most appropriate method to classify the effort and estimate the total effort. The company expressed interest in demonstrating the application and the proposed methods. In the future, the thesis topic can be followed up by a more detailed study of machine learning methods, including their design.