EKONOMICKÁ FAKULTA TUL

Bakalářská práce

Optimalizace procesů modernizace webové aplikace

Studijní program: B0688P140002 Informační management
Autor práce: **Petr Horák**
Vedoucí práce: doc. Ing. Klára Antlová, Ph.D.
Katedra informatiky

Liberec 2024

EKONOMICKÁ FAKULTA TUL

Zadání bakalářské práce

Optimalizace procesů modernizace webové aplikace

Jméno a příjmení:	**Petr Horák**
Osobní číslo:	E21000221
Studijní program:	B0688P140002 Informační management
Zadávající katedra:	Katedra informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Přehled strategií a přístupů k modernizaci softwarových řešení
2. Porovnání dostupných technologií
3. Návrh vhodné strategie pro zlepšení optimalizace procesů pomocí případové studie
4. Vyhodnocení návrhu
5. Návrh postupu implementace řešení

Rozsah grafických prací:	
Rozsah pracovní zprávy:	30 normostran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

- TANTRY, Seetharama, 2017. Implication of legacy software system modernization. *International Journal of Advance Research in Computer Science,* vol. 8, no. 7, s. 1002-1008. ISSN 0976-5697.
- BUREŠ, Miroslav; Miroslav RENDA; Michal DOLEŽEL; Peter SVOBODA a Zdeněk GRÖSSL, 2016. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada Publishing. ISBN 978-80-247-5594-6.
- DURHAM, Doug a Chad MICHEL, 2021. *Lean software systems engineering for developers: managing requirements, complexity, teams, and change like a champ*. New York: Apress. ISBN 978-1-4842-6932-9.
- POUR, Jan; Miloš MARYŠKA; Iva STANOVSKÁ a Zuzana ŠEDIVÁ, 2018. *Self service business intelligence: jak si vytvořit vlastní analytické, plánovací a reportingové aplikace*. Praha: Grada Publishing. ISBN 978-80-271-0616-5.
- SHRIVASTAVA, Saurabh; Neelanjali SRIVASTAV; Rajesh SHETH; Rohan KARMARKAR a Kamal ARORA, 2022. *Solutions Architect's Handbook*. Second edition. Birmingham: Packt Publishing. ISBN 978-1-80181-661-8.

Konzultant: Bc. Filip Svárovský, jednatel Tanganica, s.r.o.

Vedoucí práce:	doc. Ing. Klára Antlová, Ph.D. Katedra informatiky
Datum zadání práce:	1. listopadu 2023
Předpokládaný termín odevzdání:	31. srpna 2025

L.S.

doc. Ing. Aleš Kocourek, Ph.D.
děkan

Mgr. Tereza Semerádová, Ph.D.
garant studijního programu

V Liberci dne 1. listopadu 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

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

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

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

21. dubna 2024 Petr Horák

Optimalizace procesů modernizace webové aplikace

Anotace

Tato práce se zabývá problematikou modernizace informačních systémů a softwaru obecně. Vývoj softwaru je nákladná činnost. V praxi se tak často vývoj nových informačních systémů řídí předlohou v podobě již zavedených softwarových řešení. Cílem této práce je proto popsat možné přístupy k modernizaci informačních systémů a přijít s doporučeními optimalizujícími tento proces. Práce popisuje rizika tohoto procesu a obecně známé strategie modernizace softwaru. Na případové studii modernizace softwarového řešení nejmenované firmy následně přímo analyzuje tento proces v praxi.

Klíčová slova

aplikace, framework, Informační systém, marketing, modernizace, podnikové procesy, software, strategie

Optimisation of web application modernisation processes

Annotation

This thesis focuses on the modernization of information systems and software in general. Software development is an expensive activity. In practice, the development of new information systems is often guided by a template in the form of already established software solutions. Therefore, the aim of this thesis is to describe possible approaches to information systems modernization and to come up with recommendations to optimize this process. The thesis describes the risks of this process and commonly known software modernization strategies. It then analyses this process in practice using a case study of software modernization of an unnamed company.

Key Words

application, business processes, framework, information system, marketing, modernization, software, strategy

Obsah

Seznam ilustrací (obrázků)

Seznam tabulek

Seznam použitých zkratek, značek a symbolů

API	Application Programming Interface
B2B	Business to business
CI/CD	Continuous Integration and Continuous Deployment (kontinuální integrace a nasazení)
COTS	Commercial Off-The-Shelf
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
CSR	Client-side Rendering
DIČ	Daňové identifikační číslo
DOM	Document Object Model
GDPR	General Data Protection Regulation
HTML	HyperText Markup Language
IS	Informační systém
JSX	JavaScript Syntax Extension
KPI	Key Performance Indicators (klíčové ukazatele výkonu)
MVC	Model-View-Controller
ROI	Return on Investment (návratnost investice)
SAAS	Software as a Service
SPA	Single Page Application (jednostránková aplikace)
SSR	Server-side rendering

Úvod

Modernizace informačních systémů je v praxi velmi častá činnost. Málokterý podnik dokáže hned v začátcích svého působení vyvinout systém, který by mu vyhovoval po celou dobu jeho existence.

Při vývoji jakékoliv aplikace nebo softwarového řešení by měl být brán ohled na odolnost a flexibilitu daného projektu vůči změnám. Díky těmto vlastnostem je možné projekt rozvíjet snáze a delší dobu. Ať už jsou ale softwarové projekty navrženy v tomto ohledu sebelépe, každý z nich dříve nebo později bude třeba modernizovat. V závislosti na zvoleném postupu modernizace může jít v některých případech o složitý a zdlouhavý proces. Průběh, podobu a výsledek toho procesu ovlivňuje mnoho různých faktorů.

Obsah této práce je rozdělen do dvou částí. První – teoretická část práce – je věnována seznámení s problematikou modernizace softwarových řešení. Jsou zde popsány již zmíněné faktory ovlivňující proces modernizace. Následuje přehled strategií, které se při modernizaci softwaru používají. Představeny jsou také technologie, které se v dnešní době při modernizaci webových aplikací používají, a to jak z pohledu řízení procesů (např.: GitHub, Asana), tak implementace modernizovaného projektu (např.: .NET, Laravel)

Druhá – praktická část práce – obsahuje případovou studii zabývající se modernizací webové aplikace v nejmenované firmě. Daná aplikace má ve firmě roli informačního systému, proto se i tato práce nesoustředí pouze na specifika modernizace webových aplikací, ale i na problematiku vývoje informačních systémů a softwaru obecně. Jelikož je firma projektově řízena, věnuje se práce i této oblasti vývoje softwaru. Na základě poznatků studie pak práce přináší doporučení a návrh optimálního řešení modernizace v tomto daném případě.

Cílem této práce je poskytnout čtenáři jak přehled možných řešení a přístupů k modernizaci zastaralého softwaru, tak přehled doporučení k optimalizaci procesu modernizace jako takové. To vše za využití poznatků z praxe.

1 Význam modernizace informačních systémů

Pojem modernizace může v oblasti vývoje webových aplikacích označovat různé úkony a procesy. V naprosté většině případů je však společným jmenovatelem nutnost změny zastaralého sytému. Změnou můžeme chápat jak kompletní nahrazení původního sytému novým, tak pouze aktualizaci jeho částí. Jedná se o velmi nákladný proces nejen z hlediska finančních prostředků.

Podobu procesu modernizace softwarového řešení podniku značně ovlivňuje typ využívaného systému. Proces modernizace se tudíž může lišit v případě, že firma vyvíjí a spravuje vlastní softwarové řešení anebo v případě, kdy využívá tzv. SaaS (Software as a Service) nebo COTS (Commercially of the Shelf) systémů. V závislosti na tomto a mnoha dalších faktorech se může lišit jak finanční, tak časová náročnost procesu.

Modernizace softwaru je základní součástí jeho vývoje, není však důležitá jen z technologického hlediska. Vývoj systémů je v mnoha případech klíčovým prvkem strategického rozvoje podniku. Modernizace přináší zlepšení výkonnosti, zabezpečení dat, možnosti adaptace na nové technologické trendy a mnoho dalších důsledků, díky nimž může být podnik konkurenceschopnější.

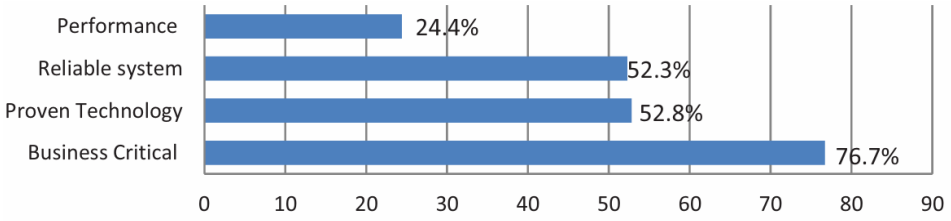
1.1 Definice zastaralého informačního systému

V oblasti neustále vyvíjejících se informačních technologií často platí, že technologie používané dnes mohou být považované za zastaralé již zítra. To však nijak nebrání provozu systémů využívající tyto technologie. Zastaralý systém lze definovat jako systém, jehož provoz je pro daný podnik kritický, avšak již ho dle požadavků nelze dále upravovat (Tantry et al., 2017).

Často lze zastaralé informační systémy najít v odvětvích jako je zdravotnictví, veřejná správa, finanční sektor nebo i v některých průmyslových podnicích (Shrivastava et al., 2022). Moderní a dobře optimalizovaný informační systém totiž nepředstavuje pro podniky působící v těchto oblastech nikterak zásadní prostředek k vykonávání jejich činnosti. Stav informačních systémů je zde tudíž často opomíjen do doby, než systém přestane fungovat.

Zastarání systému se do určité míry dá předcházet jeho údržbou. Průběžné úpravy systému, opravy chyb a aktualizace mohou značně přispět k oddálení momentu, kdy je třeba rozhodnout a jeho zásadnější modernizaci.

Dle výzkumu zabývajícího se pohledem odborníků z praxe na modernizaci zastaralých systémů je zřejmé, že i zastaralý systém má pro podnik své výhody. Téměř dvě stovky odborníků nejen z oblasti IT odpovídalo na otázky související s vnímáním, výhodami a nástrahami modernizace softwarových řešení (Khadka et al., 2014). Z tohoto výzkumu vyplynulo, jaké aspekty zastaralých systémů jsou pro podniky nejpodstatnější – viz graf číslo 1.



Obrázek 1: Výhody zastaralých IS
Zdroj: How do professionals perceive legacy systems and software modernization? (Khadka et al., 2014)

Graf 1 výše zachycuje, jaké procento respondentů vnímá jednotlivé aspekty původních systémů jako zásadní. Hlavním benefitem hovořícím pro zachování zastaralých systémů v provozu je jejich význam z hlediska provozu podniku jako takového. Případný výpadek systému pro podniky znamená nemalé ztráty. Obecně také platí, že staré systémy jsou vnímány jako ověřené a spolehlivé (Khadka et al., 2014). Bezmála čtvrtina respondentů vidí staré informační systémy jako výkonné. Z toho se dá usuzovat, že i zastaralý systém je v některých případech dostačující pro potřeby podniku z hlediska jeho výkonu a kapacity.

1.2 Důvody k modernizaci informačních systémů

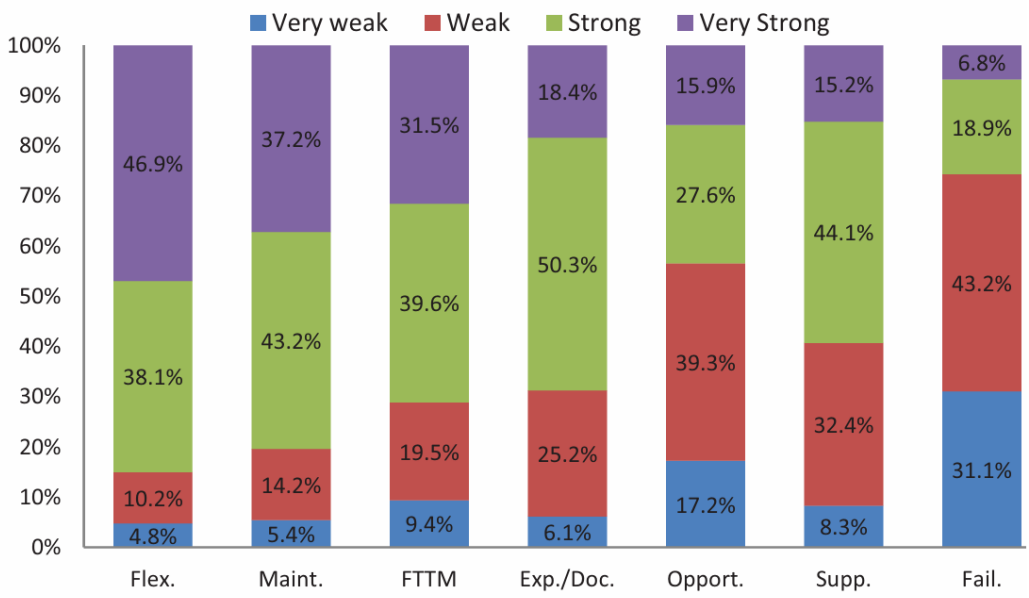
Důvodů proč modernizovat zastaralé informační systémy je mnoho. Jen některé z těchto důvodů ale bývají pro management podniků při rozhodnutí o modernizaci systému rozhodující. Pokud stávající systém z pohledu jeho běžných uživatelů funguje, nemusí být v rámci podniku pro modernizaci systému potřebná motivace. S každým dalším dnem provozu zastaralého systému se ale zvyšuje riziko výskytu problémů.

Jednoznačným důvodem pro modernizaci systému je tak například jeho častá poruchovost nebo nekompatibilita s jinými modernizovanými subsystémy. Pokud systém nelze nadále udržovat v provozu, je rozhodnutí o modernizaci zkrátka jednoznačné. Složitější rozhodování nastává v situaci, kdy úsilí a náklady vynaložené na udržování stávajícího systému v provozu přesáhnou

určitou míru. Vedení podniku, který se v takové situaci ocitne, musí zhodnotit, zda je stávající stav pro podnik únosný. Pokud ne musí najít vhodné řešení modernizace systému. Toto rozhodnutí ovlivňuje nespočet různých faktorů. Zvolení optimální strategie při modernizaci systému může eliminovat množství nástrah, které se při tomto procesu běžně objevují.

Jedním z důvodů bývá změna legislativy, kterou se musí podniky, a tudíž i jejich informační systémy řídit. Exemplárním příkladem je zavedení tzv. GDPR (General Data Protection Regulation). Toto nařízení Evropské unie se zabývá ochranou osobních údajů. Aby informační systém splňoval požadavky tohoto nařízení, musí mimo jiné poskytovat funkci k vymazání osobních údajů daného uživatele. Při vývoji systémů v dnešní době se s takovou funkcionalitou počítá od úplného začátku, některé technologie využívané k vývoji IT systémů jsou na implementaci této funkce dokonce přímo připravené. V zastaralých systémech však implementace takové funkce může představovat velmi složitý proces. V některých případech je dokonce prováděna manuálně. (Shrivastava et al., 2022)

Dle již zmíněného výzkumu je hlavním důvodem k modernizaci IS nedostatečná flexibilita a vysoké náklady na provoz zastaralých IS (Khadka et al., 2014). Z Grafu 2 lze vyčíst jednotlivé důvody k modernizaci systémů a také jakou mají váhu při rozhodování o modernizaci původního systému.



Obrázek 2: Důvody k modernizaci IS
Zdroj: How do professionals perceive legacy systems and software modernization? (Khadka et al., 2014)

Téměř polovina respondentů během výzkumu uvedla, že neflexibilní informační systém, který nereflektuje potřeby a směřování podniku, je pro ně velmi silným důvodem k modernizaci tohoto systému. Druhým nejpodstatnějším faktorem ovlivňujícím rozhodnutí podniků o modernizaci jimi

využívaných IS jsou vysoké náklady na provoz systémů. Následuje možnost rychlejšího vývoje a reakce na změny na trhu (Faster time-to-market) systému. Na čtvrtém místě je nedostatek expertních pracovníků potažmo dokumentace k zastaralému sytému, který by v případě poruchy systému zásadním způsobem narušil fungování podniku. Mezi méně podstatné faktory patří možnost integrace systému s jinými aplikacemi. Na podobné úrovni je dostupnost podpory ze strany dodavatelů technologií, které zastaralý systém používá. Na posledním místě je poněkud překvapivě poruchovost systému. Příčinou, proč není poruchovost zásadnějším faktorem by mohl být fakt, že zastaralé systémy jsou většinou ve stavu, kdy všechny chyby, které by bránili jejich provozu byly odstraněny (Khadka et al., 2014).

1.3 Proces modernizace informačního systému

Na proces modernizace se lze dívat stejnou optikou jako na proces vývoje. Zejména pokud se podnik nerozhodne modernizovat původní systém nahrazením za některý z komerčně dostupných informačních systémů, probíhá modernizace velmi podobně jako vývoj systému jako takový.

Proces lze rozdělit do tří velmi obecných kroků. Prvním z nich je **plánování**. To obnáší definici cílů modernizace a určení požadavků na modernizovaný systém. Přesně specifikované požadavky výrazně přispívají k eliminování leckterých nástrah během vývoje nové verze systému. Mimo eliminaci nástrah je dobré přesně definovat podobu a funkce modernizovaného systému také proto, aby všechny zúčastněné strany procesu měly reálné představy a očekávání od výsledného produktu. Samozřejmostí je pak také výběr optimální strategie na základě definovaných cílů. V ideálním případě se provede i zhodnocení stávajících softwarových řešení, které podnik využívá.

V případě, že je původní podnikem vyvíjený systém určitým způsobem modernizován, jedná se o klasický vývoj softwaru. Podnik je v takové situaci sám sobě zákazníkem. To zejména ve fázi plánování přináší nemalé výhody. Lze přesně definovat cíle a požadovaný konečný stav modernizovaného systému.

Dalším krokem je již samotná **implementace** postupující dle předem rozplánovaných úkonů. Ne vždy je ale možné přejít od plánování rovnou k implementaci nového řešení. Pokud modernizovaná aplikace využívá nové technologie, se kterými nejsou vývojáři sžiti, je třeba nejdříve věnovat čas jejich řádnému zaškolení a osvojení dovedností. Jestliže jsou všichni s využívanými technologiemi obeznámeni, může se začít s vývojem nového systému.

Jelikož jsou cíle a požadavky v případě modernizace systémů většinou předem a jasně definované, mohlo by se zdát, že je vhodné využít tradiční metodiky vývoje s vodopádovým životním cyklem. V kombinaci s odpovídající modernizační strategií by využití rigorózních metodik mohlo být určitým způsobem přínosné. Avšak ve valné většině případů i přes optimální podmínky pro využití těžkých metodik převládají výhody agilního řízení procesů.

Posledním krokem je **provoz** modernizované aplikace. Není ojedinělé, že se během provozu aplikace objeví nedostatky. Může jít jak o chyby v implementaci některých funkcí přímo bránící používání systému, které je třeba řešit neprodleně, tak o kosmetické úpravy a doladění detailů. Ani v této fázi procesu by se nemělo na modernizovaném systému přestat pracovat. Úsilí vynakládané na vývoj systému je v této fázi pochopitelně nižší než během implementace nového softwarového řešení, avšak i základní průběžná údržba modernizovaného sytému značně oddálí stav, kdy ho bude nutné znovu nákladně modernizovat.

1.4 Nástrahy modernizace informačních systémů

Při vývoji softwaru se lze často setkat s tvrzením, že by se nemělo měnit to, co funguje. Proto, pokud se k modernizaci nepřistoupí šetrně a pragmaticky, může se nástrah objevit více, než kdyby byla zvolena optimální strategie. Na druhou stranu, pokud z původního systému optimálně funguje pouze jeho malá část, nemusí se vyplatit snaha o zachování této části na úkor rozsáhlejších změn.

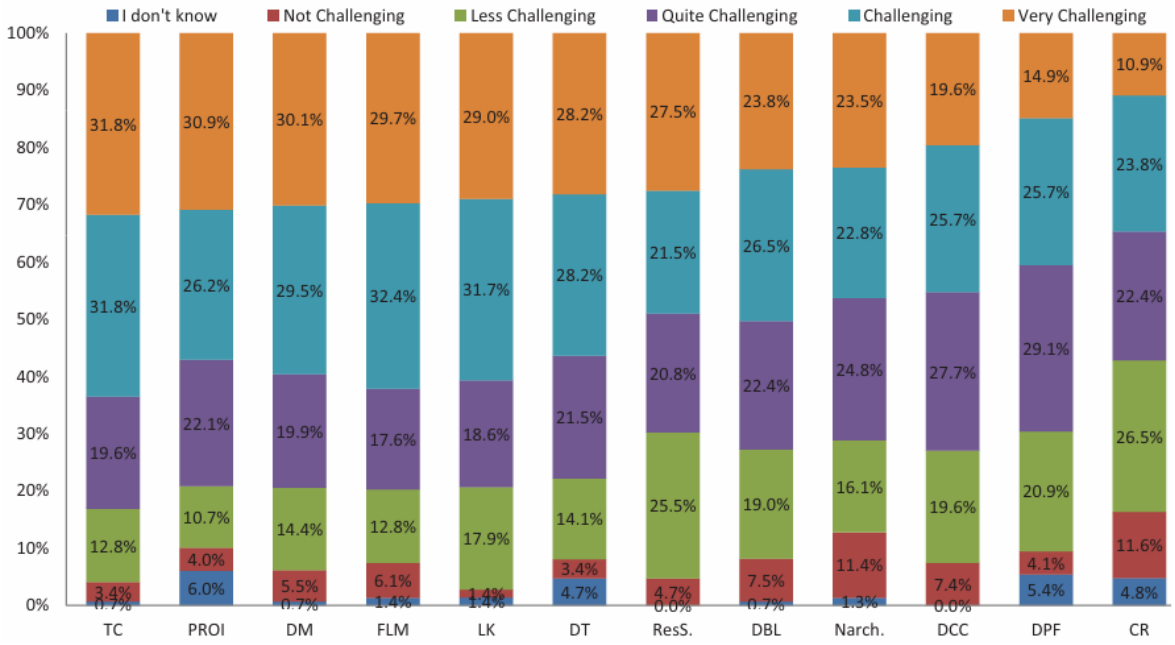
Obecně platí pravidlo, že čím je podnik větší, tím je modernizace informačního systému, který daný podnik využívá, náročnější (Tune, 2023). Velikost podniku představuje obdobnou překážku, jako rozsáhlost systému jako takového. Ve větších podnicích je více zaměstnanců, kteří jsou na původní systém a jeho funkce zvyklí než v menších podnicích. Pokud byla zvolena některá z modernizačních strategií, při níž je zastaralý systém zásadně modifikován, nebo zcela nahrazen systémem novým, nemusí se vyplatit zachovat některé funkce původního systému.

Během modernizace IS nastává ideální doba pro změnu nastavení a optimalizaci vnitřních procesů podniku, tak aby fungování podniku odpovídalo struktuře vyvíjeného systému. V menších podnicích se takové změny provádějí snáze než v těch větších. Při modernizaci IS velkých firem není od věci vznik samostatného AMET (Architecture Modernization Enabling Team) týmu, který se soustředí na eliminaci nástrah modernizace (Tune, 2023). Tento tým funguje jako spojka mezi vývojáři a

jednotlivými odděleními podniku. Při náročném a dlouho trvajícím procesu modernizace má mimo jiné za úkol udržovat zúčastněné strany motivované a obeznámené s průběhem modernizace.

Modernizace softwaru totiž nemá vliv pouze na technické a ekonomické aspekty, ale může také ovlivnit pracovní atmosféru a týmovou dynamiku. Změna využívaných technologií často znamená nové výzvy a přizpůsobení se novým pracovním postupům, což může ovlivnit náladu nejen v týmu vývojářů. Pokud se zaměstnanci cítí na změnu nepřipraveni nebo ztratí jistotu v již osvojených dovednostech, hrozí, že nebudou motivování se na vývoji nového systému podílet. V případě vývojářů samotných je třeba řešit řádné zaškolení, jinak vzniká riziko toho, že základy nového sytému nebudou implementovány optimálně a kvalitně, což může ovlivnit celý zbytek aplikace.

Výzkum vnímaní modernizace zastaralých IS se zabýval i tím, co odborníci z praxe vnímají jako největší výzvy procesu modernizace (Khadka et al., 2014). Graf číslo 3 níže pak mimo rozčlenění jednotlivých nástrah zobrazuje i za jak náročné je respondenti průzkumu považují.



Obrázek 3: Výzvy modernizace zastaralých systémů
Zdroj: How do professionals perceive legacy systems and software modernization? (Khadka et al., 2014)

Jednotlivé sloupce grafu znázorňují pohled respondentů na následující nástrahy procesu modernizace informačních systému:

- Časový tlak na dokončení modernizace (TC – Time Constraint)
- Předpovídání návratnosti investice (PROI – Predicting ROI)
- Migrace dat (DM – Data Migration)
- Financování projektu modernizace (FLM – Funding Modernization)
- Nedostatek znalostí (LK – Lack of Knowledge)
- Obtížné testování (DT – Difficult to Test)
- Odpor zaměstnanců (ResS. – Resistance from Staff)
- Obtížnost extrakce obchodní logiky (DBL – Difficult to Extract Business Logic)
- Nevyvíjitelná architektura systému (Narch. – Non-evolvable System Architecture)
- Obtížná komunikace důsledků (DCC – Difficult to Communicate Consequences)
- Obtížná prioritizace funkcionalit (DPF – Difficult to Prioritize Functionality)
- Kulturní odpor ze strany organizace (CR – Cultural Resistance)

Každá z modernizačních strategií – které jsou podrobněji analyzované níže – skýtá odlišné překážky na cestě k úspěšnému provozu modernizovaného systému.

2 Strategie modernizace softwarových řešení

Jak vyplývá z úvodu a předchozích kapitol této práce, modernizace informačního systému je velmi široký pojem. Modernizace může mít velké množství podob v závislosti na výchozím stavu původního systému a požadavcích na systém modernizovaný. Obdobně různorodé jsou i obecně známe strategie, kterými se lze při procesu modernizace systémů řídit. Strategie neboli podoba procesu modernizace, je často určena již v počátcích plánování modernizace. Volba optimální strategie totiž zásadně ovlivňuje výsledek modernizace.

2.1 Přehled strategií

Přehled níže popisuje podobu a průběh procesu modernizace za zvolení jednotlivých modernizačních strategií. Jednotlivé strategie obecně popisují možné přístupy k modernizaci softwarových řešení. Většinou tak detailně nepopisují procesy a postupy při modernizaci, ale slouží spíše jako definice přístupů k modernizaci. Ze své podstaty tak popisují proces modernizace systémů velmi obecně. V praxi není výjimkou kombinace různých přístupů.

Volbu pro podnik nevhodnější modernizační strategie ovlivňuje také velikost organizace a obor, ve kterém působí. V odvětvích, kde informační systémy představují z hlediska konkurenceschopnosti podniku zásadní faktor, je na modernizaci systémů kladen velký důraz. Oproti tomu menší organizace anebo podniky poskytující služby či produkty nesouvisející s oblastí informačních technologií zpravidla nepotřebují věnovat modernizaci systémů větší pozornost.

Následující přehled nejčastěji využívaných modernizačních strategií krátce vysvětluje každý z přístupů. Popisuje jejich výhody a podmínky, kdy je na místě zvážit jejich využití. Vysvětluje také jejich vzájemné odlišnosti. Závěrem jednotlivé strategie hodnotí.

2.1.1 Změna hostitele (Rehosting)

Změna hostitelského prostředí je jedna z méně komplexnějších strategií. Podstatou vývoje systému tímto způsobem je jeho přemístění do nového prostředí, které může mít podobu nových fyzických serverů, virtualizovaného prostředí nebo v dnešní době populárního cloudu. Zásadní je, že se

nemění funkcionalita aplikace. V rámci rehostingu dochází k minimálním změnám ve zdrojových kódech aplikace.

Z toho vyplývají i výhody a nejčastější využití této strategie. Rehosting se využívá v případech, kdy by v důsledku změn hardwarové infrastruktury, nebo například aktualizace operačního systému nemohl být informační systém nadále udržován v provozu. Tato strategie se osvědčuje při migraci velkých a komplexních systémů, kde jsou klíčové minimalizace rizik a zachování stability systému. Podniku může tento způsob modernizace snížit náklady na provoz systému beze změny fungování systému.

Narozdíl od ostatních strategií jako je revitalizace nebo replacement nabízí rehosting rychlou implementaci a tím i nižší náklady. Ne vždy je ale vhodné tuto strategii zvolit, často může jen oddálit nutnost zásadnějších změn v implementaci funkcí aplikace. To zejména v případě, že modernizovaný systém není optimalizovaný a má slabá místa. Určitou výhodu oproti replatformingu či refaktoringu je fakt, že se nezabývá přímými úpravami v kódu aplikace ani její interní strukturou.

Při zhodnocení efektivity této strategie velmi záleží na konkrétních případech využití. V závislosti na konkrétních potřebách a cílech podniku může být rehosting efektivním a pragmatickým přístupem k modernizaci. V jiných případech – obzvlášť pokud není stávající softwarové řešení podniku kvalitní – nemusí přinést kýžený výsledek.

2.1.2 Nahrazení (Replacement)

Název této strategie zcela vystihuje její podstatu. Původní nevyhovující systém je zcela nahrazen nově vyvinutým systémem. Zastaralý systém s modernizovaným nesdílí žádné funkční části. Často je tento přístup k modernizaci spojen se zásadnějšími změnami v nastavených podnikových procesech. Tantry ho popisuje jako tzv. "big bang approach" kvůli často velmi vysokým nákladům na vývoj kompletně nového systému (Tantry et al., 2017).

Tento přístup se vyplatí využít pouze v případech, kdy je původní softwarové řešení natolik neflexibilní, že jej nelze dále vyvíjet ani modernizovat některou z jiných strategií tak, aby systém splňoval požadavky podniku. Největší výhodou toho přístupu je možnost vytvořit zcela nový systém s využitím moderních technologií a architektur. Nový systém může být navržen tak, aby lépe

vyhovoval budoucím potřebám a mohl být snadněji vyvíjen v závislosti na rostoucích požadavcích podniku.

Existuje také možnost nahradit zastaralý systém některým z pro daný podnik vhodných COTS Saas systémů. V takovém případě záleží na dodavateli systému, jaké možnosti v rámci přechodu na nový systém poskytuje. Jelikož jsou prvky procesu modernizace v takovém případě závislé na konkrétních případech a využitích, nelze obecně určit přínosy a rizika takového řešení.

V porovnání s rehostingem se nahrazení nachází na opačném konci spektra náročnosti provedení modernizace. Vývoj zcela nového systému přináší mnoho rizik. Mimo jiné se uživatelé musí seznámit s odlišnostmi nového systému, který musí být řádně otestován. Obecně lze říct, že se od ostatních přístupů odlišuje hlavně kompletním nahrazením zastaralého systému, který je odstaven a nadále se již žádná z jeho částí nevyužívá.

Rozhodnutí o využití této strategie by mělo předcházet zvážení ostatních možností, neboť náklady na implementaci jsou oproti jiným řešením modernizace vysoké a dosažení očekávaných výsledků není zaručeno. V kombinaci s dobrým plánováním eliminujícím nástrahy na cestě k spuštění nového systému může tento přístup zlepšit konkurenceschopnost podniku.

2.1.3 Revitalizace (Revitalization)

Revitalizace softwarových řešení znamená postupné vylepšování a odstraňování slabých částí systému. Nejedná se tedy o nijak radikální řešení, ale o kontinuální vývoj zaměřený na již implementované nevyhovující části systému. Díky průběžné revitalizaci systému lze úspěšně předcházet situacím, kdy je třeba systém modernizovat využitím některých z radikálnějších a nákladnějších strategií.

S revitalizací systému je vhodné začít ještě v době, kdy je stávající systém stále funkční, ale začíná vykazovat některé nedostatky. Včasná identifikace těchto nedostatků a rozhodnutí o jejich odstranění může pro podnik znamenat zachování investic do existujícího systému, což snižuje riziko a náklady spojené s migrací na zcela nový systém. Pokud je však stávající systém natolik neflexibilní, že jeho postupné úpravy jsou z dlouhodobého hlediska stejně nákladné jako nahrazení systému novým, je na místě zvolit radikálnější přístup k modernizaci.

Oproti ostatním strategiím je revitalizace konzervativnějším přístupem k modernizaci. Revitalizace má podobné rysy jako refaktoring. Na rozdíl od refaktoringu, který se soustředí na úpravy kódu a vnitřní struktury systému, revitalizace zahrnuje širší škálu úprav a aktualizací, včetně aktualizace uživatelského rozhraní, optimalizace výkonu a migrace dat.

V závislosti na konkrétních vlastnostech původního systému tak může být revitalizace efektivním a pragmatickým přístupem k modernizaci bez větších narušení provozu systému či rizik. Pokud ale stávající systém obsahuje závady, které brání jeho provozu, nebo jeho struktura neodpovídá potřebám podniku, nemusí být pouhá revitalizace daného systému vhodným řešením modernizace.

2.1.4 Změna platformy (Replatform)

Podstata této strategie je podobná jako u rehostingu s tím rozdílem, že se nemění prostředí ale technologická platforma, kterou stávající systém využívá. Změny se tak odehrávají na nižší úrovni a počítá se i se změnami v kódu. Tento přístup může zahrnovat aktualizaci operačního systému, databázového systému, softwarových frameworků nebo dalších technologických komponent, které stávající systém využívá. Můžeme se setkat také s označením retargeting, které má v souvislosti s modernizací informačních systému stejný význam.

Důvody pro využití této strategie jsou většinou technického rázu. Tím je myšlena například zastaralá verze operačního systému, nedostatečná škálovatelnost nebo kompatibilita s novými technologiemi. Aktuální je využití replatformingu v souvislosti s přechodem systémů na cloudové technologie. Modernizace systému tímto způsobem přináší řadu výhod. Aktualizace technologické platformy může zvýšit výkon, bezpečnost i spolehlivost systému.

Nejblíže replatformingu je reshosting, v rámci kterého se také nijak nemění funkce systému. Při změně platformy jsou však potřeba změny v kódu, což s sebou nese zvýšené náklady a časovou náročnost. Oproti rehostingu je pak třeba také rozsáhlejší testování modernizovaného systému.

Změna platformy patří mezi méně využívané modernizační strategie. Pádné důvody k jejímu využití totiž nenastávají často. Hlavním přínosem replatformigu je zejména zvýšení výkonu systému, nikoliv možnost implementace nových funkcí, což pro některé podniky nemusí být atraktivní.

2.1.5 Refaktoring (Refactoring)

Refaktoring je mezi vývojáři pravděpodobně nejznámější pojem z dosud uvedených. Zabývá se úpravami a optimalizací existujícího kódu a interní struktury aplikace. Cílem je zlepšit čitelnost, udržitelnost a efektivitu kódu beze změn jeho vnější funkcionality. To znamená odstranění nedostatků v kódu jako jsou duplicity, zbytečně komplexní části, neefektivní algoritmy nebo špatně navržené datové struktury.

Aby mohl být systém vyvíjen efektivně, je určitá míra refaktoringu nutná. Častou překážkou pro refaktorování kódu je neochota ze strany managementu podniku věnovat takové činnosti, která zdánlivě nepřináší žádnou přidanou hodnotu, cenný čas vývojářů. Z dlouhodobého hlediska je takový pohled na věc mylný. Náklady na další vývoj nerefaktorovaného systému mohou být i několikanásobně vyšší, než na vývoj systému s dobře organizovanými a čitelnými zdrojovými kódy. To značně usnadňuje a zefektivňuje práci vývojářů, což může být kritický faktor pro konkurenceschopnost podniku.

Oproti některým strategiím se modernizace s využitím refaktoringu zaměřuje pouze na technologický aspekt systému. Na rozdíl od revitalizace, která se zaměřuje na přepracování a vylepšení celého systému, se refaktoring soustředí výhradně na úpravy kódu a interní struktury systému. Podobně však pomáhá předcházet nutnosti modernizovat stávající systém některým z nákladnějších přístupů, jako je například nahrazení systému.

V ideálním případě by měl refaktoring probíhat průběžně s vývojem systému. V případě, že byl systém vyvíjen rychle anebo nezkušenými vývojáři, je na místě kompletní refaktoring systému. Kompletní refaktoring přináší výhody zejména v oblasti čitelnosti kódu, ale také zvýšení výkonu a responzivity systému. Důsledkem toho je zefektivnění vývoje nových funkcí.

2.1.6 Změna architektury (Re-architecting)

Jedná se o další ze strategií, které se zaměřují z větší části na technologický aspekt systému. Funkce modernizovaného systému by se nijak zásadně neměly lišit od systému původního. Změna architektury znamená přepracování klíčových komponent systému, jako jsou například datové modely nebo struktura databáze. Z hlediska implementace se tak jedná o náročný proces, který přináší značná rizika.

Často se lze se změnou architektury setkat v případě, kdy stávající systém využívá monolitickou architekturu a podnik je stále větším rozsahem systému nucen přejít na architekturu mikro služeb. Systém vyvíjený jako monolitický se vyznačuje tím, že je nasazen na jednom serveru a v případě jakýchkoliv změn je třeba sestavení a nasazení celého systému. Oproti tomu rozdělení systému do tzv. mikro služeb znamená, že je systém složený z na sobě nezávislých menších aplikací, které spolu navzájem komunikují skrze API. To přináší řadu výhod jako je modularita a zpřehlednění systému, snadnější implementace nových funkcí nebo snazší škálovatelnost a odolnost systému.

Po modernizaci nahrazením zastaralého systému jde v případě změny architektury o něco méně rozsáhlé řešení modernizace, ale v porovnání s většinou ostatních přístupů ji lze považovat za velmi rozsáhlou a implementačně náročnou. Vývojáři musí být pečlivě seznámeni jak se stávající, tak s novou architekturou, což zvyšuje požadavky na jejich znalosti ve srovnání s ostatními modernizačními strategiemi.

V případě správné implementace změna architektury přináší zásadní zlepšení systému s ohledem na jeho budoucí vývoj. Zejména v případě již zmíněného přechodu od monolitu k využití mikro služeb můžou mít přínosy nové architektury výrazný vliv na konkurenceschopnost podniku.

2.1.7 Přepsání (Rewrite)

Podstata této strategie tkví ve využití moderních technologií k vytvoření nového systému. Jak název napovídá, zdrojový kód zastaralého systému je přepsán často do jiného jazyka využívajícího modernější technologie. Jde o radikální řešení modernizace, při kterém se můžou změnit některé funkce systému.

Zvolit tento přístup k modernizaci informačního systému je vhodné v případě, že stávající systém vyhovuje z hlediska podnikových procesů, avšak je natolik zastaralý, že již například nesplňuje bezpečnostní požadavky. Příkladem využití může být přepsání starého systému napsaného v zastaralém programovacím jazyce do modernějšího jazyka, jako je například migrace aplikace z jazyka COBOL do jazyka Java. Podobný přechod je z hlediska dlouhodobého vývoje systému nutný a přináší podniku nespočet výhod.

Svojí podstatou se tento přístup blíží řešení modernizace nahrazením původního systému. V rámci nahrazení se však může změnit i struktura databáze, nebo některé podnikové procesy týkající se práce s informačním systémem. Oproti strategiím jako je refaktoring nebo změna architektury,

které se zaměřují na úpravy a optimalizaci stávajícího systému, představuje přepsání radikálnější přístup a umožňuje vytvoření nového základu pro budoucí vývoj.

V případě že struktura původního systému splňuje požadavky podniku, ale systém nevyhovuje v jiných oblastech, je vhodné modernizovat systém tímto způsobem. Ve většině případů je ale na místě zkombinovat přepsání s úpravami ve struktuře databáze a úpravami dalších komponent systému. Tím už se ale dostáváme od pouhého přepsání k nahrazení.

2.2 Další strategie

Mezi méně známé modernizační strategie patří například revamping. Jedná se o strategii, která se zaměřuje zejména na uživatelské rozhraní systému. Hlavním přínosem modernizace tímto způsobem je tak zpříjemnění práce uživatelů se systémem. Výhody tohoto přístupu však můžou snadno převážit jeho nevýhody. Systém modernizovaný tímto způsobem totiž z pravidla nikdy není výkonnější či responzivnější než původní systém.

Podobně netradiční je i metoda nazývaná jako wrapping. Původní systém je obalen nadstavbou, která umožňuje integraci s modernějšími technologiemi a systémy. Funkčnost původního systému zůstává z velké části stejná. Ve většině případů je tento přístup využit jako přechodné řešení, neboť základ systému modernizovaného tímto způsobem se nemění a může tak v budoucnu představovat slabé místo systému.

Lze se setkat také s pojmem reinženýring. Ve své podstatě jde o nahrazení původního systému nově vyvinutým systémem. Dá se říct, že jde o stejný přístup k modernizaci jako u nahrazení.

3 Nástroje pro vývoj informačních systémů

V oblasti vývoje webových aplikací se lze setkat s mnoha různými technologiemi. Tato kapitola se proto bude soustředit jen na ty aktuálně populární a hojně využívané. Kapitola je rozdělena do dvou částí. První část se zaobírá nástroji pro řízení procesů vývoje a druhá je věnována představení aktuálně nejpopulárnějších frameworků a technologií pro vývoj webových aplikací jako takový.

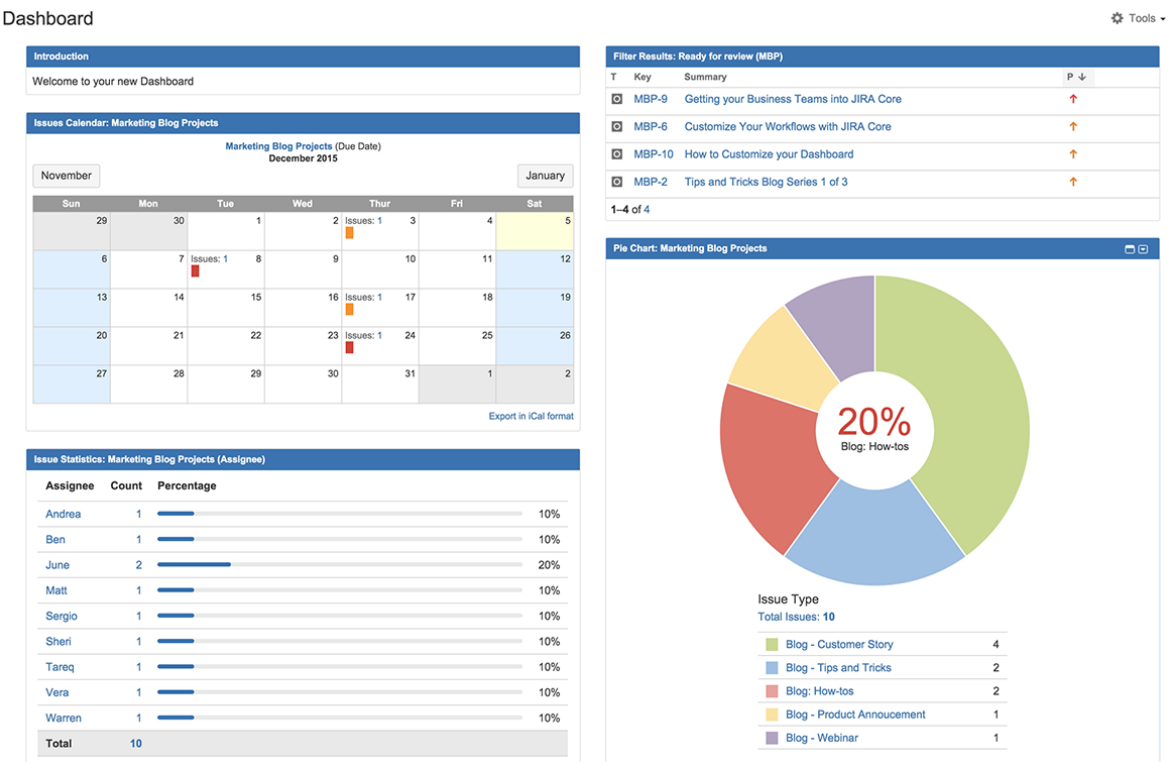
3.1 Projekt management nástroje

Kvalita výsledného produktu většinou závisí na řízením projektu. S využitím stále populárnějších agilních metodik vývoje je třeba využívat i tomu odpovídající nástroje pro správu daného projektu. Od manažerů se vyžaduje, aby pochopili průběh projektu a kvalitu produktu bez potřeby rozsáhlých vývojových dokumentů. Proto se při agilním vývoji často používají nástroje pro řízení projektů. Používání těchto nástrojů vede k rychlejšímu a efektivnějšímu vývoji a ovlivňuje kvalitu výsledného softwaru. (Özkan a Mishra, 2019)

Následující přehled krátce představuje populární nástroje, které organizace při vývoji softwaru často používají. Některé nejsou primárně určeny pro projekt management, ale obsahují funkce, díky kterým můžou k tomuto účelu také sloužit.

Jira

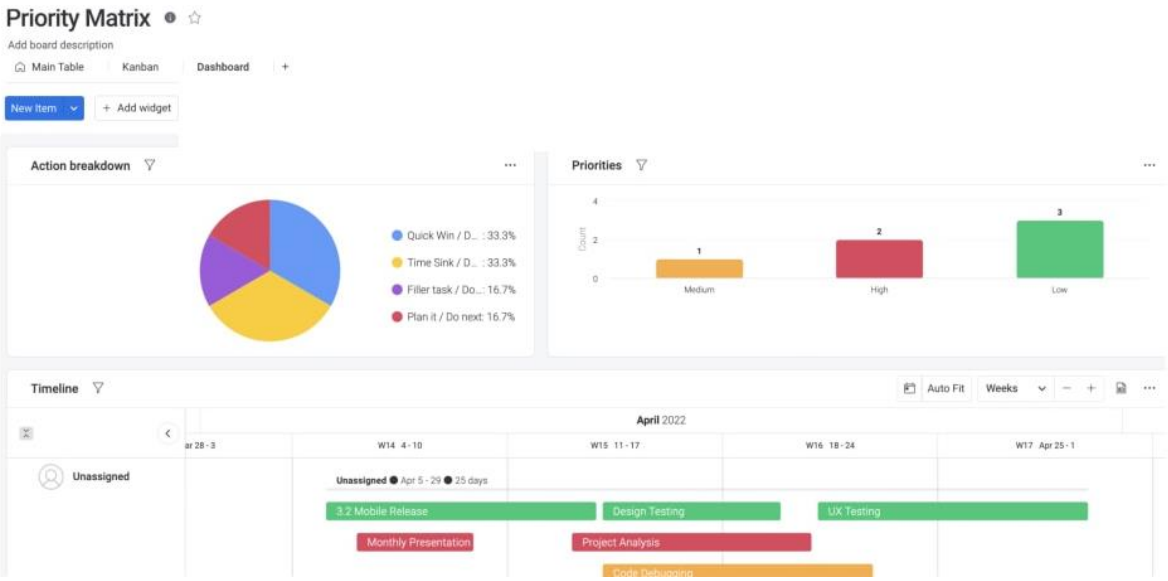
Jira je populární nástroj pro řízení projektů, jehož uživatelské rozhraní je přizpůsobeno architekturám vývoje jako je Scrum nebo Kanban. Nabízí integraci s ostatními nástroji, mezi které patří například GitHub. Mezi další důležité vlastnosti Jira lze zařadit přizpůsobitelné integrace vývojářských nástrojů, kumulativní vývojové diagramy, reportování nebo sledování problémů a chyb. Vzhledem k tomu, že se jedná o vyspělý a osvědčený nástroj, který preferují velké podniky po celém světě, má tento nástroj aktivní komunitu uživatelů (Özkan a Mishra, 2019).



Obrázek 4: Dashboard aplikace Jira
Zdroj: www.atlassian.com

Monday.com

Tato platforma pomáhá týmům při vývoji softwaru efektivně řídit projekty pomocí nástrojů pro sledování času, reportování a týmovou spolupráci. Jejími přednostmi jsou zejména možnosti správy pracovních postupů a automatizace. Umožňuje vytvořit vlastní pracovní postupy, které budou odpovídat procesu vývoje produktu. Dále lze automatizovat klíčové procesy, aby vše probíhalo hladce a snížilo se manuální zatížení týmu. Automatizovat lze například přiřazování úkolů, oznámení či integraci pracovních postupů s dalšími nástroji. (Hermann, 2023)



Obrázek 5: Dashboard aplikace monday.com
Zdroj: www.moday.com

Asana

Tato platforma má standardní funkce pro správu projektů, jako je vytváření úkolů, dílčích úkolů a přizpůsobení ovládacích panelů. Pomáhá usnadňovat spolupráci mezi jednotlivými odděleními, která je ve vývojových týmech zásadní. Jeden úkol lze přiřadit více projektům, což pomáhá vyhnout se duplicitním záznamům a zpřesňuje vykazování odvedených činností. Cíle lze nastavit na více úrovních, což usnadňuje pochopení toho, jak každý úkol přispívá k dokončení celého projektu. Lze také sledovat pracovní vytížení zaměstnanců a týmů, aby bylo zajištěno, že nikdo nebude přepracovaný. (Hermann, 2023)

Obrázek 6: Dashboard aplikace Asana
Zdroj: www.asana.com

GitHub

GitHub je nejen platforma pro správu verzí a spolupráci v softwarovém vývoji, ale stává se také nástrojem pro řízení projektů. Umožňuje týmům efektivně sdílet kód, spolupracovat na jeho vývoji a řešit problémy prostřednictvím vytváření tzv. issues a pull requestů. Ty se pak dají přehledně zobrazit a filtrovat, nebo přiřadit jednotlivým vývojářům. Integrace s nástroji jako Jira nebo Slack umožňuje propojení GitHubu s dalšími nástroji pro správu projektů a komunikaci v týmu. Za nevýhodu GitHubu se dá považovat komplexní a složité prostředí pro uživatele, kteří nejsou obeznámeni s používáním verzovacích systémů.

GitLab

GitLab je podobně jako GitHub platforma pro správu verzí a spolupráci v softwarovém vývoji, ale s důrazem na integraci celého softwarového vývoje do jednoho systému. Jednou z jeho hlavních předností je možnost spravovat systém na vlastním serveru, což umožňuje větší kontrolu nad daty a infrastrukturou. Obdobně jako GitHub nabízí GitLab širokou škálu funkcí pro řízení projektů, jako jsou nástěnky, issues a CI/CD (Continuous Integration/Continuous Deployment). Mezi nevýhody lze zahrnout složitější nastavení spojené se správou vlastního serveru.

3.2 Technologie webových aplikací

Tato kapitola je rozdělena do dvou částí. V první části jsou představeny aktuálně populární technologie v oblasti vývoje frontend části aplikací. Tyto technologie se soustředí na zdokonalení uživatelského rozhraní a jeho vývoje. Druhá část se zabývá možnostmi pro vývoj backend části aplikací. Technologie zde využité obstarávají funkcionalitu systémů od autorizace uživatelů po spojení s databází.

3.2.1 Frontend technologie

Frontend technologie typicky využívají jazyky jako HTML, CSS a JavaScript, které definují strukturu, styl a chování webových stránek. V praxi se ale většinou pouze kombinace těchto jazyků k vývoji webových aplikací nepoužívá. Využívají se různé knihovny a frameworky, které usnadňují a zrychlují vývoj. Mezi ně patří i React, Vue.js a Svelte.

React

React je JavaScript knihovna, kterou vydal Facebook v roce 2013. React se zaměřuje zejména na vytváření tzv. single-page aplikací, jejichž vývoj usnadňuje hlavně prostřednictvím komponent, které jsou znovupoužitelné a izolované. Komponenty jsou základními stavebními kameny aplikace, které přijímají vstupní data a vrací HTML šablony jako výstup. React používá virtuální DOM (Document Object Model), který je rychlejší a efektivnější než skutečný DOM v prohlížeči. React také podporuje JSX, což je rozšíření JavaScriptu, které umožňuje používat HTML syntaxi uvnitř JavaScriptu. (Kaluža et al., 2018)

Vue.js

Vue je framework pro vytváření uživatelských rozhraní, který vznikl v roce 2016. Vue je komplexní a výkonný framework pro vývoj interaktivních uživatelských rozhraní. Stejně jako React, Vue je založen na komponentách, které jsou znovupoužitelné. Vue používá také virtuální DOM, který zlepšuje rychlost a efektivitu zobrazování dat. Vue je dnes jeden z nejrychleji rostoucích JavaScriptových frameworků. Vue je vhodný pro vývoj jak single-page aplikací, tak multi-page aplikací. Jelikož se jedná o framework, nikoliv o knihovnu jako React, není pro práci s Vue nutná hluboká znalost JavaScriptu. (Kaluža et al., 2018)

Svelte

Svelte je moderní JavaScript framework vytvořený v roce 2016. Na rozdíl od React a Vue Svelte nemá virtuální DOM a místo toho přímo manipuluje s reálným DOM. Svelte se zaměřuje na jednoduchost a rychlost. Jeho syntaxe je čistá a jednoduchá, což usnadňuje vývoj a údržbu kódu. Kód napsaný ve Svelte je kompilován do malých a optimalizovaných balíčků. To činí Svelte ideálním pro vývoj výkonných webových aplikací. Jeho nevýhodou může být fakt, že je stále relativně nový. (Libby, 2022)

3.2.2 Backend technologie

Backend označuje serverové části webových aplikací, které definují logiku a funkcionalitu webových aplikací. Využívají se zde jazyky jako C#, PHP, Java, Python a další. I zde se využívají převážně frameworky postavené na některém z těchto jazyků. Hojně využívanými jsou Microsoft .NET, Laravel a Flask.

.NET

Microsoft .NET je framework, který umožňuje vytvářet webové aplikace v jazycích C# a VB.NET. Nabízí širokou škálu nástrojů a knihoven usnadňující práci s daty, grafikou, autorizací uživatelů, testováním a dalšími aspekty vývoje (Vermeir, 2022). Díky tomu je vývoj aplikací pro vývojáře seznámené s různorodými komponenty tohoto frameworku vcelku jednoduchý. Prvotní seznámení však nemusí být jednoduché, vzhledem ke složité konfiguraci projektu, místy nepřehledné dokumentaci a integraci s ostatními Microsoft technologiemi. Výhodou je jeho popularita a dostupnost neoficiální dokumentace.

Laravel

Laravel je framework pro vývoj webových stránek, který využívá programovací jazyk PHP. Podobně jako .NET je Laravel postaven na architektuře MVC (Model-View-Controller) (Yadav et al., 2019). Díky tomu je základní princip vývoje aplikací s využitím obou technologií podobný. Dle Richarda (Richard, 2022) je jednou z hlavních výhod oproti .NET jednoduchost počáteční konfigurace projektu. Oficiální dokumentace je také o něco přehlednější, strádá však dokumentace komunitní, neboť je .NET populárnější. Leckdy lze považovat za výhodu i absenci určitého nátlaku na využití příbuzných technologií, jako je tomu v případě .NET a Microsoft. Na druhou stranu je Laravel oproti .NET méně robustní.

Flask

Flask je minimalistický webový framework využívající jazyk Python. Zaměřuje se na poskytování základních funkcí bez vynucování specifické architektury projektu, díky čemuž je vysoce přizpůsobitelný. Primárně vychází z architektury MVC, nechává však na vývojářích, aby ji implementovali podle svých potřeb. Přestože se nejedná striktně o MVC framework, jeho flexibilita umožňuje vývojářům vytvářet jak jednoduché aplikace, tak rozsáhlé a složité projekty. Flask vyniká ve scénářích, kdy vývojáři preferují implementaci vlastních architektonických řešení. Tím umožňuje efektivně vytvářet webové aplikace na míru. (Maia, 2015)

4 Modernizace informačního systému startupu

Tato kapitola formou případové studie popisuje proces modernizace vlastního softwarové řešení nejmenované firmy. Mimo samotný proces modernizace jako takové je popsán i počáteční stav původního řešení a hlavní důvody k jeho modernizaci. Následně je analyzován stav po nasazení modernizovaného systému, jeho přínosy a vliv na fungování podniku.

Firma je na trhu přibližně tři roky a má okolo patnácti stálých zaměstnanců. Lze ji tak stále označit jako startup. Hlavní činností firmy je poskytování marketingových služeb svým zákazníkům, jimiž jsou nejčastěji správci internetových obchodů. K tomu slouží firmou vyvíjená webová aplikace, ke které mají přístup i samotní zákazníci. Formu podnikání tak lze označit jako B2B SaaS.

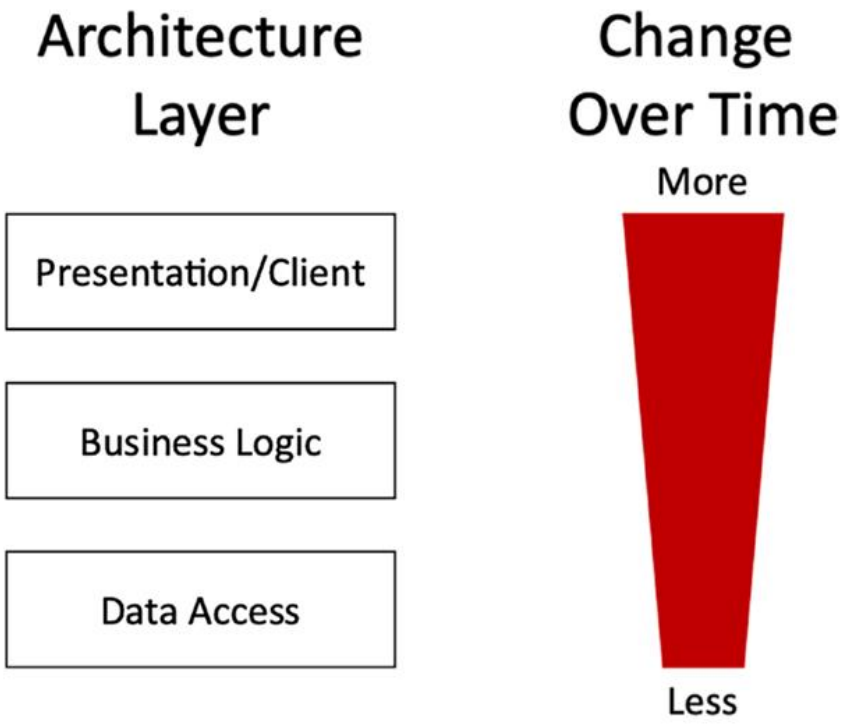
I přes to, že do aplikace mají přístup samotní zákazníci, lze ji zároveň považovat za plnohodnotný firemní informační systém. Společně s aplikací dostupnou zákazníkům tvoří zbytek tohoto systému také samostatné správcovské prostředí. To v některých ohledech slouží jako CRM systém (Customer Relationship Management), ale zároveň také umožňuje správu několika dalších oblastí podstatných z hlediska podnikového řízení. Mezi ně patří například nástroje pro analýzu klíčových ukazatelů výkonnosti (tzv. KPI – Key Performance Indicator), modul pro práci se systémem generovanými účetními doklady nebo nespočet funkcí pro správu profilů zákazníků v aplikaci.

Základními požadavky na systém z pohledu efektivního fungování celého podniku je jeho dostatečná robustnost a flexibilita. Tyto v některých ohledech možná protichůdné požadavky vychází z nutnosti propojení systému s mnoha externími systémy. Systém tak musí být odolný vůči neočekávaným scénářům vyvolaným změnou chování připojených systémů třetích stran. V případě výskytu problémů je žádoucí možnost co nejjednoduššího procesu nápravy problému.

4.1 Původní řešení

Firma se od počátku svého působení soustředí na jednoduchost a rychlost v implementaci nových řešení. To v začátcích umožnilo rychlý postup kupředu a snadné přidávání nových funkcí do fungujícího systému. Vzhledem k situaci v nově vzniklém startupu, nebylo reálné přesně definovat požadavky na systém. To, v kombinaci s tehdy nezkušenými členy oddělení vývoje, znamenalo převážnou absenci pevných základů celého systému.

Při návrhu původního řešení se věnovala pozornost z velké části pouze softwarovému vývoji, a nikoliv softwarovému inženýrství. Rozdíl mezi těmito dvěma pojmy vysvětluje Durham. Softwarové inženýrství definuje jako disciplinovaný přístup k tvorbě softwaru se zaměřením na zásady a postupy, které vedou ke kvalitnímu softwaru. Vývoj se v jeho pohledu zabývá přímo zdrojovým kódem. (Durham a Michel, 2021)



Obrázek 7: Četnost změn v hlavních částech systému
Zdroj: Lean software systems engineering for developers (Durham a Michel, 2021)

Kvůli nedostačujícímu návrhu původního systému bylo často nutné dělat změny i v části systému obstarávající získávání dat z databáze. Nezřídka kdy se měnila i struktura a konfigurace databáze samotné. Standardní stav četnosti změn napříč hlavními částmi systémů znázorňuje grafika na obrázku 7. Z grafiky lze vyčíst, že v dobře navržených systémech se nejčastěji mění jejich uživatelské rozhraní. Choulostivější části systému jako je část implementující byznys logiku podniku nebo databázová část systému, by se měly měnit méně často. Durham nicméně podotýká, že změny v systému jsou znakem jeho aktivního využití. Vzápětí však zmiňuje důležitost poměru času stráveného s údržbou a re-implementací na úkor času věnovanému vývoji nových funkcí (Durham a Michel, 2021). Tento poměr se v případě původního systému dotyčné firmy časem vymkl kontrole a implementace nových funkcí způsobovala čím dál větší obtíže.

Přístup firmy k vývoji systému se dá popsat takzvaně jako frontend-first. To označuje zaměření vývoje hlavně na uživatelské rozhraní systému a tvorbu dobrého dojmu uživatelů při používání systému. Uživatel tak na první pohled vidí dobře fungující systém, zatímco v pozadí je vyžadováno velké množství i manuálních úkonů, aby byly splněny akce, které uživatel při práci se systémem vyvolal. Tento model se ukázal být výhodný v počátcích působení firmy díky nízkému počtu klientů. S narůstajícím počtem zákazníků byla nutná automatizace manuálních procesů. Ta přinesla eliminaci lidských chyb a snížení nákladů na provoz.

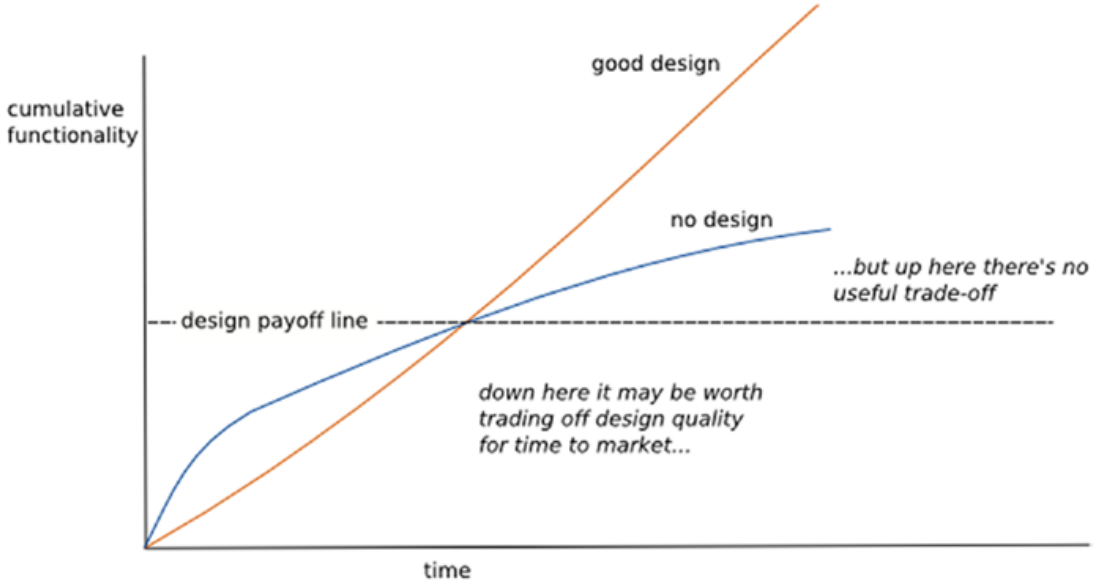
I když je zmiňovaný frontend-first přístup často software vývojáři a programátory zatracovaný, stojí za zmínku jeho využití nejen v malých a začínajících podnicích. Určitou podobnost k tomuto přístupu můžeme spatřovat v oblasti dnes velmi populární umělé inteligence. I společnost jako je Amazon přiznala, že některé funkce označované jako využívající nástrojů umělé inteligence ve skutečnosti využívají spíše lidských zdrojů z Indie (Chowdhury, 2024). I přes to, že vývoj informačních systémů tímto stylem rozhodně není ideální, jedná se o nezřídka kdy využívaný přístup.

Přes všechny své nedostatky, umožnilo původní řešení systému firmě vyrůst a dostat se do bodu, kdy bylo možné vyvinout vyspělejší systém využívající moderní technologie.

4.1.1 Backend technologie

Zdrojové kódy backend části původního systému byly psány téměř výhradně v jazyce PHP. A to bez využití jakéhokoliv z frameworků. Veškerá logika aplikace tak byla rozložená do mnoha skriptů a funkcí implementovaných přímo v nativním PHP. Tato skutečnost v kombinaci se z počátku méně zkušeným týmem vývojářů předznamenala vznik nespočtu neoptimálních řešení, která negativně ovlivnila pozdější vývoj systému.

Absence frameworku s sebou nemusí nést vždy jen samé nevýhody. V ojedinělých případech lze při programování pouze s využitím daného programovacího jazyka bez jakýchkoliv nadstaveb dosáhnout lepších výsledků. Lepších nejčastěji s ohledem na optimalizaci výkonu, práci s pamětí a dalších aspektů ovlivňujících efektivitu informačních systémů. Tento styl vývoje vývojářům dává volný prostor, a výsledný systém tak může být velmi flexibilní a přímo uzpůsobený potřebám podniku bez výraznějších technologických omezení.



Obrázek 8: Hypotéza výdrže návrhu systému
Zdroj: Lean software systems engineering for developers (Durham a Michel, 2021)

Podobný fenomén popisuje Durham grafem na obrázku výše. Z grafu lze vyčíst, že tým vyvíjející software bez určitého návrhu a plánování může být zpočátku rychlejší a efektivnější, ale nakonec – a poměrně rychle – je tým, který při vývoji postupuje podle kvalitního návrhu, překoná. A to nikoli zvýšením jejich vlastní rychlosti, ale dramatickým poklesem rychlosti týmu vyvíjejícím software bez kvalitního návrhu. (Durham a Michel, 2021)

Volba postupu upřednostňujícího rychlost a tzv. „time to market“ se při vývoji původního systému ukázala být chybným rozhodnutím, neboť již zmíněná flexibilita, kterou vývoj tímto stylem nabízí, vyústila v nekontrolované „bobtnání“ projektu. Bez předem a jasně stanovených vzorů a pravidel, které nabízí vývoj s využitím frameworku, byla orientace v rozsáhlých zdrojových kódech aplikace postupem času stále obtížnější. A to zejména pro nové členy oddělení vývoje, což představovalo zásadní problém z hlediska rozvoje podniku.

Samostatnou otázkou v případě vývoje bez frameworku poskytujícího knihovny pro implementaci základních funkcí webových aplikací je bezpečnost. Obecně – a v oblasti bezpečnosti obzvlášť – platí, že není vždy výhodné přicházet s vlastní implementací standardizovaných a leckdy i volně dostupných řešení. I přes nemalé úsilí věnované právě bezpečnosti a zabezpečení, byla řešení této problematiky v původním systému přinejmenším nevyhovující.

V pozdějších fázích vývoje původního systému byly některé jeho nové funkce implementovány v jazyce Python. Jednalo se ale hlavně o okrajové části systému obsluhující propojení s aplikačními rozhraními třetích stran.

4.1.2 Databáze

Za nejslabší část původního systému lze považovat právě jeho relační MySQL databázi. I její stav totiž odráží styl vývoje bez předem jasně daných pravidel a zásad. Mimo ne zcela optimální strukturu relací nebyla výjimkou ani absence základních integritních omezení, či nestálá přesnost některých datových typů. To vše jde proti postupům, které popisuje Pour. Ten zdůrazňuje, že databáze by měly odpovídat aktuálním, ale i budoucím požadavkům. Mají také být relativně bezpečně rozšiřitelné a udržovatelné. Případné úpravy databáze nesmí znamenat nepřiměřená rizika pro kvalitu dat. (Pour et al., 2018)

4.1.3 Frontend technologie

Jak již bylo zmíněno, část systému definující podobu uživatelského rozhraní byla od počátku vývoje aplikace věnována o něco větší pozornost. Postupný vývoj v této oblasti dokázal velmi dobře maskovat technické nedostatky logické části systému v pozadí. I zde však nastaly situace, kdy stav backend části systému představoval přítěž pro rychlý a efektivní vývoj aplikace.

Na rozdíl od backendu byla část aplikace definující podobu a rozvržení uživatelského rozhraní systému modernizována již v době provozu původního systému. Na úplném počátku byla frontend část systému vyvíjena podobným stylem jako backend část. Tím je myšleno převážné využití čistě základních technologií jako jsou jazyky HTML a CSS. Uživatelské rozhraní tak bylo velmi statické, protože byl obsah stránek aplikace generován pomocí PHP na straně serveru. To byl hlavní z důvodů k rozhodnutí o modernizaci této části systému, neboť vývoj takto rozsáhlého systému s využitím této architektury je v mnoha ohledech nevhodný.

Krátkozrakost budování systému tímto stylem vedla k první zásadnější modernizaci softwarového řešení firmy. Frontend část klientské části aplikace byla nově implementována s využitím frameworku Vue.js. Aby taková transformace byla možná, bylo třeba zásadních úprav v architektuře backend části systému. Ta stále využívala nativní PHP, avšak to mělo nyní architekturu klasického API. Tento změnou frontend technologií vyvolaný přerod s sebou přinesl další komplexitu

vycházející ze zcela vlastní implementace struktury API. Mnohem větší pozornost tak musela být věnována zabezpečení a mnoha dalším aspektům takto rozvrženého systému. Tato změna se však týkala pouze klientské části systému, správcovská část přístupná pouze zaměstnancům firmy nikterak propracované a interaktivní uživatelské rozhraní nepotřebovala.

Ve výsledku tak z části modernizovaný systém kombinoval oba přístupy. Klientská část aplikace využívala tzv. CSR (client-side rendering) s využitím Vue.js a správcovská část fungovala na principu SSR (server-side rendering). Tato skutečnost nijak nepřispěla zpřehlednění struktury celého projektu a komplikovala tak mimo jiné začlenění nových vývojářů do procesu vývoje systému. Na druhou stranu přispěla tato proměna ke zlepšení dojmu z aplikace jako takové, která působila profesionálnějším dojmem. Firma tak byla schopna oslovit více zákazníků.

4.2 Proces modernizace

Postupem času a s přibývajícími funkcemi systému bylo jasné, že určitou formou modernizace bude muset projít i backend část systému. Žalostný stav původního řešení jistým způsobem usnadnil volbu strategie, jakou byl systém modernizován. Zcela adekvátním přístupem bylo kompletní nahrazení systému novým. Žádná z ostatních méně radikálních strategií vzhledem k chatrným základům původního systému nebyla vhodná. Pouhé úpravy a modifikace systému byly vyhodnoceny jako neefektivní a příliš nákladné.

4.2.1 Plánování

Hlavním mottem provázejícím celý proces modernizace tak v reakci na stav původního systému byla co možná nejvyšší míra standardizace a využití obecně známých technologií a postupů. Záměrem za tímto přístupem bylo co možná nejvýznamnější oddálení situace, která nastala při vývoji původního systému. Chatrné základy systému zde negativně ovlivnily následující vývoj, který byl neefektivní a plný technologií vynucených kompromisů. Cílem modernizace proto bylo vytvoření zcela nového systému, který splňuje potřeby podniku a je připravený na intenzivní vývoj v dalších letech fungování firmy.

Po volbě strategie modernizace následoval výběr technologií a konceptuální návrh databáze. Mezi zvažovanými backend technologiemi byly Node.js, Laravel a Microsoft .NET. První z nich byla vyřazena z důvodu složitosti, a tudíž časové náročnosti implementace celého systému (Mardan,

2018). Využití PHP frameworku Laravel se nabízelo už kvůli určité návaznosti na předchozí systém, ale zejména také kvůli určité kompatibilitě s Vue.js, které využívala frontend část aplikace. Finální volba však padla na balík technologií Microsoft .NET. Rozhodující byla jeho popularita mezi vývojáři a umožnění rychlého vývoje některých základních částí systému, díky dostupným knihovnám (Jackson, 2020). Podobný přístup provázel i volbu databázového systému, kdy díky své popularitě, přehlednosti a široké nabídce funkcí zvítězilo PostgreSQL (Juba et al., 2015).

I když byly požadavky na nový systém přesně vymezeny funkcemi původního systému, nebylo vhodné se nevyhovující předlohou při návrhu základů nového systému nikterak řídit. Prvotní konceptuální návrh databáze tak v průběhu modernizace prošel mnoha menšími i většími změnami, neboť se i přes dostupnou předlohu v podobě původního systému nepodařilo přijít s návrhem optimální struktury databáze, kterou by se v následující fázi intenzivního vývoje dalo řídit. Počáteční návrh tak definoval zejména základy fungování systému. Mimo jiné přicházel s definicí implementace tzv. multi-tenant architektury, která obstarává oddělení dat jednotlivých klientů v databázi. Vzhledem k tomu, že firma vyvíjený systém nabízí svým zákazníkům formou SaaS byla optimální implementace této architektury kritická.

4.2.2 Implementace

Před začátkem prací na samotném vývoji nového systému a implementací jeho funkcí tak byly známy a analyzovány technologie, které budou využity. Byla nastíněna základní struktura databáze, což částečně určilo i základní konfiguraci systému. K dispozici byla také dokumentace původního systému, která podrobně popisovala strukturu API, vstupy a výstupy jednotlivých koncových bodů a zdroje, ke kterým každá akce přistupuje.

Vznik takto detailní dokumentace byl zčásti zapříčiněn původním plánem provozu původního a modernizovaného systému zároveň. Zamýšlena totiž byla postupná modernizace, kdy by byl původní systém nahrazován po částech. To by však znamenalo udržovat v provozu oba systémy v určité symbióze. Hlavní překážkou při tom byla nutnost modernizace databáze, a tudíž změna její struktury. Rizika a komplexita takového postupu značně předčila jeho výhody. Během průběhu celého vývoje nového systému byl původní systém udržován v provozu až do nasazení modernizovaného systému.

V úplných počátcích na vývoji nového systému pracoval pouze jeden člen oddělení vývoje. Další se věnovali nejen údržbě, ale stále také i implementaci nových funkcí do původního systému. Bylo tak nereálné dostihnout vývoj původního systému. Řešení této situace spočívalo v rozšíření týmu vývojářů zabývajícím se modernizací systému v kombinaci s omezením vývoje původního systému. Většina nyní čtyřčlenného týmu věnujícího se vývoji backend části nového systému s projekty podobného formátu a rozměrů neměla zkušenost, bylo tak nutné věnovat čas jak jejich seznámení s problematikou samotného vývoje, tak seznámení s funkcemi původního systému.

Po zvážení všech výše zmíněných faktorů – jako je rozšíření týmu, dostupnost dokumentace, sestavený konceptuální návrh databáze a výběr technologií – hovořil realistický odhad dokončení modernizace a spuštění nového systému o čtyřech až pěti měsících. To včetně zaškolení nových členů týmu. V případě přenesení přesné struktury původního API do nového systému, jak bylo původně zamýšleno, by byl tento termín pravděpodobně splněn. Nedlouho po začátku intenzivního vývoje nového systému se však zachování struktury původního API ukázalo být značně limitujícím faktorem a nevyhovujícím řešením.

Změna struktury API přispěla i k přenesení veškeré logiky aplikace na backend. V původním systému se totiž nedostatky logické části aplikace kompenzovaly implementací logiky do frontend části systému. Zásadním způsobem se proto měnilo i fungování části aplikace definující vlastnosti uživatelského rozhraní. Veškeré změny však byly motivovány standardizací a zpřehledněním struktury celého projektu. Struktura nového API byla navržena tak, aby odpovídala rozložení a komponentům uživatelského rozhraní aplikace.

Během změn struktury API a během procesu modernizace obecně se ukázalo být vhodné implementovat některé nové funkce systému. Rozhodnutí o nemodernizování částí původního systému, které měly být v dohledné době nahrazeny novými funkcemi systému, bylo strategickým rozhodnutím. Rozdíl mezi strategickým a taktickým vývojem softwaru popisuje Durham. Taktický vývoj se zaměřuje na krátkodobé cíle, což často vede k rychlým řešením bez ohledu na dlouhodobé důsledky, zatímco strategický vývoj klade důraz na dlouhodobou perspektivu a upřednostňuje udržitelný a přizpůsobivý návrh systému, který umožňuje budoucí růst a změny (Durham a Michel, 2021). Taktický přístup by velel k modernizaci všech částí původního systému, tak jak bylo původně plánováno, což by v důsledku znamenalo neohrožení stanoveného termínu. Byl ale zvolen strategický přístup a v rámci modernizace byly rovnou vyvinuty funkce, které podle původního plánu měly být implementovány až po nasazení nového systému.

Zavedení změn a tím i odstranění nelogičností původního systému znamenalo změnu některých základních funkcí systému. Rozsáhlost procesu modernizace tak nabrala další rozměr. Jednodušší a efektivnější totiž bylo změnit některé podnikové procesy, jejichž podoba byla určena nyní pozměněnými funkcemi původního systému. Modernizace se tak v důsledku dočkala i samotná tzv. byznys logika a fungování podniku.

Nejznatelnější změny se v tomto ohledu vyskytly v implementaci a funkčnosti správcovského prostředí. Tato část systému totiž od počátku fungování původního systému neprošla jakoukoliv zásadní modernizací a implementována byla čistě pomocí PHP a HTML. To narozdíl od klientské části aplikace znamenalo absenci jakékoliv struktury připomínající API. K dispozici zde tak nebyla dokumentace. Při návrhu struktury a implementaci části nového systému obsluhující správcovské prostředí se postupovalo pouze podle předem modernizovaného a restrukturalizovaného uživatelského rozhraní.

Zásadní odlišností oproti původnímu systému, která zapříčinila změny v klientské i správcovské části systému, bylo zacházení se systémem generovanými účetními doklady. Požadavkem na nový systém totiž bylo zavedení vystavování zálohových faktur v případě, že klient zvolí platbu převodem. Finální faktura je tak správně vystavena až při potvrzení platby klientem zvolené částky. Společně se změnami v logice vystavování faktur došlo i ke změně poskytovatele platební brány pro platby kartou. Takto významné změny v nejchoulostivější části systému si vyžádaly důkladné testování jak v průběhu vývoje, tak při finálním testování před uvedením nového systému do provozu.

Došlo i k řadě menších změn napříč celým systémem, které neměly zásadní vliv na základní funkčnost systému. Většina těchto změn byla motivována již zmíněnou standardizací a zjednodušením systému. Vycházely tak z nastavení určitých pravidel v systému. Jedním příkladem za všechny, který vcelku dobře demonstruje prosazovaný přístup k modernizaci, je nastavení validace DIČ klientů registrujících se do systému. Původní systém kontroloval pouze formát klientem zadaného DIČ, nikoliv však jeho platnost. Často tak při následném generování faktur nastávaly komplikace a bylo často nutné faktury přegenerovávat se správnými fakturačními údaji klientů. Nový systém se takovým situacím snaží předcházet kontrolou platnosti zadaného DIČ. V systému je tak pravidlem, že veškerá uložená DIČ lze považovat za platná, což usnadňuje rozhodování o tom, zda má být klientovi účtována daň či nikoliv.

Samostatnou kapitolou během implementace nového systému byla migrace dat z původní databáze do nové. I přes snahu aplikovat přístup využití co nejjednodušších a nejstandardnějších

řešení využívaný napříč zbytkem projektu i při implementaci skriptů obstarávajících migraci databáze, se tato část projektu stala problematickou. Žalostný stav původní databáze, její nepřehledná struktura, absence základních integritních omezení a nestálost dat využití tohoto přístupu komplikovala. Řešení migrace dat do nové databáze ale bylo nutné, neboť jeden ze základních požadavků na modernizovaný systém byla dostupnost historických dat i z nového systému. Tato skutečnost představovala překážku při změnách v rámci vývoje struktury nové databáze, kdy při každé úpravě či zásahu do struktury nové databáze musel být brán ohled na vliv těchto úprav na migraci dat z původní databáze.

4.2.3 Provoz systému

Ostrému provozu systému předcházelo rozsáhlé testování obou aplikací – jak správcovské, tak klientské části. Postupovalo se podle předem definovaných scénářů, které byly zaměřeny obzvláště na nejkomplexnější části systému. Potvrdilo se Burešovo tvrzení, že testování představuje průměrně 20 % až 40 % pracnosti v projektech vývoje softwaru. Pro vývoj řídicího softwaru s vysokými požadavky na spolehlivost může pracnost testování často i přesáhnout pracnost vývoje. (Bureš et al., 2016)

Nasazení nového systému obnášelo mnoho předem rozplánovaných úkonů. Kvůli migraci dat z původní databáze do nové bylo před tímto procesem nutné odstavit původní systém, tak aby do původní databáze nebyly během migrace zapisovány další nová data. Právě migrace dat představovala nejkritičtější a nejkomplexnější část celého procesu spuštění nového systému. Proces migrace dat obnášel spuštění předem připravených výpočetně náročných skriptů ve správném pořadí, tak aby bylo možné migrovaná data v nové databázi řádně navzájem spojit pomocí cizích klíčů, a aby byla splněna i další integritní omezení. Migrace byla během vývoje nového sytému několikrát testována. Vzhledem k časové náročnosti však nebylo možné otestovat celý proces, ale pouze jeho části.

Odhadovaná časová náročnost spuštění nového systému se pohybovala okolo osmi hodin. Na odstávku původního a spuštění nového systému tak byla vyhrazena sobota, kdy je aplikace obvykle méně vytížena. I kvůli zdánlivě nevýznamným změnám těsně před ostrým pokusem o migraci dat a spuštění systému nebyl první pokus úspěšný. Nová databáze neobsahovala všechna data. Úspěšný byl až druhý pokus. I tak bylo ale třeba manuálně řešit některé menší nesrovnalosti. Tyto i další problémy znamenaly výrazné protažení celého procesu, a to ve výsledku na celý víkend.

Následný provoz systému se v prvních dnech neobešel bez menších problémů. Základní funkce systému se ale osvědčily a bylo třeba řešit pouze nedostatky, které přímo neovlivňovaly základní funkčnost systému. Nejčastější překážkou, na kterou klienti při přístupu do nového systému naráželi, byla nutnost nastavení nového hesla. Z důvodu bezpečnosti totiž nebylo možné přenést i tzv. hash hesla jednotlivých uživatelů do nového systému. I přes upozornění uživatelů na tuto skutečnost a poskytnutí jasných instrukcí k nastavení nového hesla, působil tento proces některým klientům obtíže.

4.3 Výsledek modernizace

Výsledkem několikaměsíčního snažení oddělení vývoje a mnoha dalších zaměstnanců podniku byl nový informační systém odrážející jak aktuální potřeby firmy, tak i ty předpokládané budoucí. Odolnosti projektu vůči budoucím změnám bylo dosaženo stanovením pevných základů systému. Úsilí vložené do maximální standardizace implementovaných funkcí znamenalo mimo zefektivnění vývoje nových funkcí také zpřehlednění struktury systému. Jedním z neméně podstatných důsledku toho je jednodušší začlenění nových vývojářů do vývoje systému.

I přes náročnost celého procesu a nemalé náklady vynaložené na modernizaci systému se investice do vývoje nového systému vyplatila. Zvýšila se konkurenceschopnost firmy a snížily se relativní náklady na vývoj. Firma je prostřednictvím modernizovaného systému schopna nabízet kvalitnější služby pro své zákazníky. Systém již není omezujícím faktorem při rozvoji firmy. Výrazně se totiž zvýšila kapacita systému a zlepšilo se zabezpečení celého systému i dat jeho uživatelů.

Modernizace rozhodně neprobíhala hladce. O výskytu mnoha překážek a nejasností vypovídá konečná délka trvání celého procesu. I když se původně očekávalo spuštění nového sytému do maximálně pěti měsíců, vyžádal si vývoj nového systému ve finále měsíců devět. Důvody k takto zásadnímu nedodržení stanoveného harmonogramu se zabývá následující kapitola. Přináší také návrh možných opatření, jejichž implementace do procesu by s velmi vysokou pravděpodobností modernizaci systému zefektivnila a umožnila by přiblížení odhadovaného k reálnému termínu dokončení.

4.4 Návrh optimálního řešení modernizace

Vývoj nového systému striktně nedodržoval žádnou ze známých metodik vývoje. Vzhledem k měnícím se požadavkům na systém a implementaci nových funkcí v průběhu vývoje nového řešení měl ale průběh vývoje blíže k agilním metodikám nežli k těm rigorózním. Tomu nasvědčovalo také rozdělení povinností a organizace práce v poměrně malém týmu, kdy se dařilo využít individualit a silných stránek jednotlivých členů. Kooperace a sdílení znalostí v týmu bylo klíčové a mělo pozitivní vliv na kvalitu nového systému.

Agilní vývoj ovšem skýtá nespočet omezení, která ovlivnila i vývoj dotyčného systému. Omezení a nástrahy vývoje softwaru s využitím agilních metodik popisuje Turk (Turk et al., 2014). Nežádoucí efekt těchto omezení bylo umocněn již zmíněným neřízením se žádnou z definovaných agilních metodik, ale spíše jejich veskrze nahodilou kombinací. Neboť se jednalo o vývoj kompletně celého softwarového řešení firmy, limitujícím faktorem byla hlavně omezená podpora agilních metodik pro vývoj kritických, rozsáhlých a komplexních softwarových řešení.

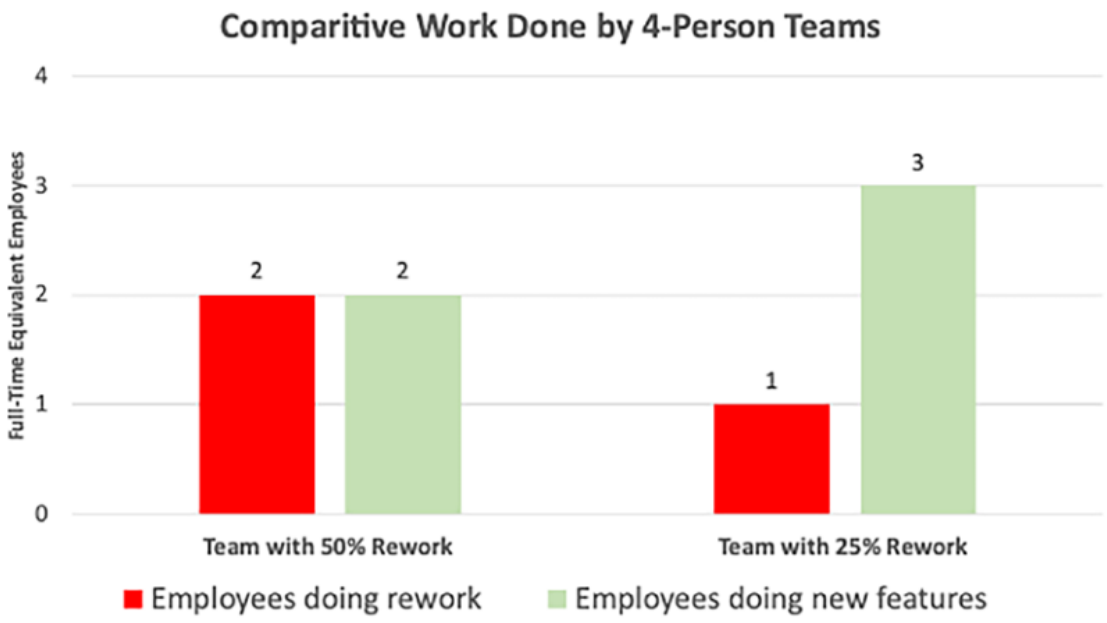
Těmto nedostatkům se dá předcházet tzv. modelováním. Využití modelování při agilním vývoji softwaru popisuje Karagiannis. Charakteristikami agilních metod modelování jsou dle něj adaptabilita, rozšiřitelnost, integrovatelnost, operativnost a použitelnost agilního modelování v reakci na měnící se požadavky na modelované řešení (Karagiannis, 2015). Předem rozvržený postup vývoje umožňuje lépe pochopit podstatu vyvíjeného řešení a zefektivňuje komunikaci v týmu.

4.4.1 Kvalitní návrh systému

V návaznosti na výše zmíněné agilní modelování se patří zmínit, že přílišné úsilí vložené do návrhu systému a podrobnému popisu jeho funkcí rozhodně není žádoucí. V tomto případě se však jednalo o modernizaci existujícího systému, který sloužil jako předloha a určitým způsobem definoval minimální požadavky na systém nový. K optimalizaci celého procesu modernizace by proto přispěla důkladnější součinnost týmu zabývajícím se vývojem a správou původního systému s týmem vyvíjejícím systém nový. Bližší spoluprací obou týmů zejména na počátku fáze plánování procesu modernizace by bylo možné dojít k například o mnoho podrobnějšímu nežli jen konceptuálnímu návrhu databáze.

Stejnou příčinu měla i další překážka ve vývoji nového systému. Původní návrh totiž počítal s vývojem systému funkcemi a do jisté míry strukturou odpovídajícímu původnímu řešení. Kvalitnější model systému by s největší pravděpodobností odhalil neprozíravost takového vývoje. Ve výsledku byla modernizace systému spojena s vývojem a implementací některých nových funkcí, což bylo strategické rozhodnutí zefektivňující z dlouhodobého hlediska vývoj celého systému. Taková změna však měla vliv na podobu celého procesu modernizace a mimo jiné také na termín dokončení. Kvalitnější definice celého procesu by proto přispěla ke stanovení reálnějšího termínu. Množství rizika a nejistoty, které je skryto kvůli nedostatečnému společnému porozumění požadavkům a očekáváním popisuje Durham (Durham a Michel, 2021).

Nedokonalý návrh vedl zejména v začátcích intenzivního vývoje nového systému k přílišné inspiraci původním řešením a přenesení některých jeho nedostatků i do nové aplikace. Části systému obsahující takové nedostatky musely být ještě během modernizace adekvátně předělány. Durham se na grafu níže zabývá vlivem předělávání již implementovaných funkcí na kumulativní pracovní sílu čtyřčlenného týmu vývojářů.



Obrázek 9: Dopad předělávání na čtyřčlenný tým
Zdroj: Lean software systems engineering for developers (Durham a Michel, 2021)

Z grafu výše vyplývá, že přesné a všemi stranami pochopené zadání je klíčové pro efektivní vývoj, neboť eliminuje riziko potřeby reimplementace již implementovaných funkcí. Proces modernizace softwarového řešení dotyčné firmy by tak mohl být významným způsobem optimalizován předem přesněji definovaným zadáním.

4.4.2 Změna podnikových procesů

Kvalitně navrhnout systém bez předlohy v podobě jakéhokoliv původního řešení představuje o mnoho obtížnější úkol. Návrh původního systému v počátcích fungování firmy tak nepočítal s mnoha později definovanými požadavky. Vývoj původního systému tak byl plný kompromisů a nevyhovujících řešení. Jelikož se dá provoz systému považovat za hlavní činnost firmy, většina podnikových procesů se odvíjela od funkcí systému. S postupem času a s ustálením firmy na trhu vykrystalizovala i podoba nabízených služeb.

Modernizace a s ní spojený vývoj nového systému představoval ideální příležitost pro zjednodušení některých procesů pokřivených nevyhovujícím fungování původního systému. Překážkou tomuto zjednodušení ale byla jistá forma „profesní deformace“ a návyku zaměstnanců na zavedené postupy. Inspirace původním systémem byla patrná zejména v modernizovaném uživatelském rozhraní správcovské části systému. Zadáním k implementaci funkcí této části systému byla právě pouze požadovaná podoba jeho uživatelského rozhraní. Vzhledem k tomu, že právě fungování správcovského prostředí systému určuje i podobu mnoha podnikových procesů, bylo na místě věnovat jeho návrhu větší úsilí.

Mnoho změn a vylepšení bylo iniciováno samotným oddělením vývoje. Přizpůsobovat a ohýbat architekturu nového systému návykům jeho uživatelů na zbytečně komplikované funkce původního systému totiž bylo náročnější než danou funkci implementovat jednoduše a přímočaře. Optimálním řešením podobných situací při takto rozsáhlých změnách informačních systémů proto může být i úprava zastaralých podnikových procesů.

4.4.3 Úprava byznys pravidel

Podobně jako změnu podnikových procesů, může být výhodné v rámci modernizace upravit i tzv. byznys logiku samotného systému a nastavená pravidla. V případě dotyčné firmy se takových pravidel měnilo hned několik. Respektive se pravidla ve většině případech spíše zaváděla, než měnila. Původní systém totiž implementoval minimum zásad, neboť bylo nutné vyhovět rozmanitým potřebám zákazníků. Striktně nastavená pravidla by totiž v úvodu fungování firmy neumožňovala akvizici nových klientů potřebnou pro růst firmy. V případě potřeby nestandardního zacházení s jednotlivými klienty byl proto do původního systému implementován nespočet výjimek obcházejících byznys logiku systému.

V době vývoje nového systému už měla firma dostatek klientů využívajících jejich služeb a další pravidelně přibývali. Zároveň již existovalo oddělení péče o zákazníky. Bylo tak možné v novém systému nastavit restriktivnější pravidla. To znamenalo určité převedení povinností, či stanovení požadavků, na zákazníky. Restriktivnější povaha systému umožnila odstranění spletitých výjimek napříč systémem a vedla k jeho celkovému zjednodušení. V rámci modernizace softwarových řešení podniků proto může být výhodné vzít v úvahu i aktuální pozici firmy na trhu a upravit nastavená byznys pravidla systému.

Závěr

Předmětem této práce bylo analyzovat možné přístupy k modernizaci softwarových řešení podniků se zaměřením na webové aplikace. Teoretická část proto byla věnována analýze této problematiky s využitím odpovídajících a aktuálních zdrojů. Přinesla také popis a porovnání známých strategií využívaných při modernizaci softwaru. Byly představeny aktuálně populární technologie pro vývoj webových aplikací, a to jak z hlediska řízení procesů, tak implementace řešení.

Praktická část popisovala rozsáhlý proces modernizace softwarového řešení malého podniku. Tento podnik modernizovaný systém přímo využívá k poskytování služeb svým zákazníkům. Vývoj nového systému měl proto pro podnik zásadní význam a kvalitní výsledek modernizace byl klíčový pro budoucí rozvoj podniku. Výstupy této případové studie v podobě návrhů na optimalizaci daného procesu jsou proto relevantní a přenositelné k aplikaci ve vývoji podobných projektů.

Spojení poznatků vycházejících z obou částí práce nabízí jedinečný pohled na řešené téma. Kombinace teorie s poznatky z praxe vycházejících z provedené případové studie ukazuje, že optimalizace procesů modernizace informačních systémů není pouze technickou záležitostí, ale zahrnuje i strategické rozhodování s ohledem na fungování celého podniku. Klíčovým faktorem je důkladný návrh řešení, který odráží potřeby podniku a jeho zákazníků. Radikální změny ve struktuře a fungování informačních systémů vyžadují systematický přístup zohledňující prostředí, ve kterém podnik působí.

Správná volba strategie modernizace informačního systému je z hlediska náročnosti a nákladnosti celého procesu naprosto zásadní. Kvalitní návrh řešení modernizace je neméně důležitý. Umožňuje sjednotit požadavky a očekávání od modernizovaného řešení, což zpřehledňuje a zjednodušuje vývoj. Při radikálním řešení modernizace by měla být věnována pozornost nejen modernizovanému systému, ale také dalším aspektům fungování podniku.

Proces modernizace informačního systému představuje pro podniky nelehký úkol. Jeho efektivní řešení s kvalitním výstupem však může znamenat zásadní zvýšení konkurenceschopnosti podniku.

Seznam použité literatury

BUREŠ, Miroslav; Miroslav RENDA; Michal DOLEŽEL; Peter SVOBODA; Zdeněk. GRÖSSL et al., 2016. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. První vydání. Praha: Grada. ISBN 978-80-247-5594-6.

DURHAM, Doug a Chad MICHEL, 2021. *Lean software systems engineering for developers: managing requirements, complexity, teams, and change like a champ*. New York: Apress. ISBN 978-1-4842-6932-9.

HERMANN, Fink, 2023. 10 Best Software Development Project Management Tools For 2024. online. In: *The Digital Project Manager*. Dostupné z: https://thedigitalprojectmanager.com/tools/best-project-management-tools-for-software-development/. [citováno 2024-02-17].

CHOWDHURY, Hasan, 2024. Amazon's „Just Walk Out" pullback shows AI has a long way to go. online. In: *Business Insider*. Dostupné z: https://www.businessinsider.com/amazons-just-walk-out-pullback-shows-ai-way-to-go. [citováno 2024-04-06].

JACKSON, Les, 2020. *The Complete ASP.NET Core 3 API Tutorial: Hands-On Building, Testing, and Deploying*. online. Berkeley, CA: Apress. Dostupné z: https://doi.org/10.1007/978-1-4842-6255-9.

JUBA, Salahaldin; Achim VANNAHME a Andrey VOLKOV, 2015. *Learning PostgreSQL*. Packt Publishing Ltd. ISBN 978-1-78398-919-5.

KALUŽA, Marin; Krešimir TROSKOT a Bernard VUKELIĆ, 2018. COMPARISON OF FRONT-END FRAMEWORKS FOR WEB APPLICATIONS DEVELOPMENT. online. *Zbornik Veleučilišta u Rijeci*, roč. 6, č. 1, s. 261–282. Dostupné z: https://doi.org/10.31784/zvr.6.1.19.

KARAGIANNIS, Dimitris, 2015. Agile modeling method engineering. online. In: *PCI '15: 19th Panhellenic Conference on Informatics*. Athens Greece, 10. 2015. ACM. Dostupné z: https://doi.org/10.1145/2801948.2802040.

KHADKA, Ravi; Belfrit V. BATLAJERY; Amir M. SAEIDI; Slinger JANSEN a Jurriaan HAGE, 2014. How do professionals perceive legacy systems and software modernization? online. In: *ICSE '14: 36th International Conference on Software Engineering*. Hyderabad India, 31. 5. 2014. ACM. Dostupné z: https://doi.org/10.1145/2568225.2568318.

LIBBY, Alex, 2022. *Practical Svelte: Create Performant Applications with the Svelte Component Framework*. online. Berkeley, CA: Apress. Dostupné z: https://doi.org/10.1007/978-1-4842-7374-6.

MAIA, Italo, 2015. *Building web applications with Flask: use Python and Flask to build amazing web applications, just the way you want them!*. 1. ed. Birmingham, Mumbai: Packt Publishing. ISBN 978-1-78439-615-2.

MARDAN, Azat, 2018. *Practical Node.js: Building Real-World Scalable Web Apps*. online. Berkeley, CA: Apress. Dostupné z: https://doi.org/10.1007/978-1-4842-3039-8.

ÖZKAN, Deniz a Alok MISHRA, 2019. Agile Project Management Tools: A Brief Comprative View. online. *Cybernetics and Information Technologies*, roč. 19, č. 4, s. 17–25. Dostupné z: https://doi.org/10.2478/cait-2019-0033.

POUR, Jan; Miloš. MARYŠKA; Iva STANOVSKÁ a Zuzana ŠEDIVÁ, 2018. *Self service business intelligence: jak si vytvořit vlastní analytické, plánovací a reportingové aplikace*. První vydání. Praha: Grada Publishing. ISBN 978-80-271-0616-5.

RICHARD, Guy, 2022. A Framework Comparison: .NET and Laravel. online. roč. 22. Dostupné z: https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1503&context=honorstheses.

SHRIVASTAVA, Saurabh; Neelanjali SRIVASTAV; Rajesh SHETH; Rohan KARMARKAR a Kamal ARORA, 2022. *Solutions Architect's Handbook - Second Edition*. Second edition. Packt Publishing. ISBN 978-1-80181-661-8.

TANTRY, H. Seetharama; N. N. MURULIDHAR a K. CHANDRASEKARAN, 2017. Implications of Legacy software system modernization - a Survey in a Changed Scenario. online. *International Journal of Advanced Research in Computer Science*, roč. 8, č. 7. Dostupné z: https://www.proquest.com/docview/1931126568/abstract/7ABEC1590A604ADEPQ/1.

TUNE, Nick, 2023. Architecture Modernization: Aligning Software, Strategy, and Structure. online. In: *Copenhagen Developer's Festival*. Copenhagen, 1. 9. 2023. Dostupné z: https://www.youtube.com/watch?v=Ls3XnV3BIyU.

TURK, Dan; Robert FRANCE a Bernhard RUMPE, 2014. Limitations of Agile Software Processes. 2014-09-22.

VERMEIR, Nico, 2022. *Introducing .NET 6: Getting Started with Blazor, MAUI, Windows App SDK, Desktop Development, and Containers*. New York: Apress. ISBN 978-1-4842-7318-0.

YADAV, Neha; Dharmveer Singh RAJPOOT a Shri Krishna DHAKAD, 2019. LARAVEL: A PHP Framework for E-Commerce Website. online. In: *2019 Fifth International Conference on Image Information Processing (ICIIP)*, 11. 2019. Dostupné z: https://doi.org/10.1109/ICIIP47207.2019.8985771.